

ELF52 - Sistemas Microcontrolados

Linguagem Assembly - Introdução, Memória e Operações Lógicas

Professor:

Prof. Marcos Eduardo

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Linguagem de Máquina

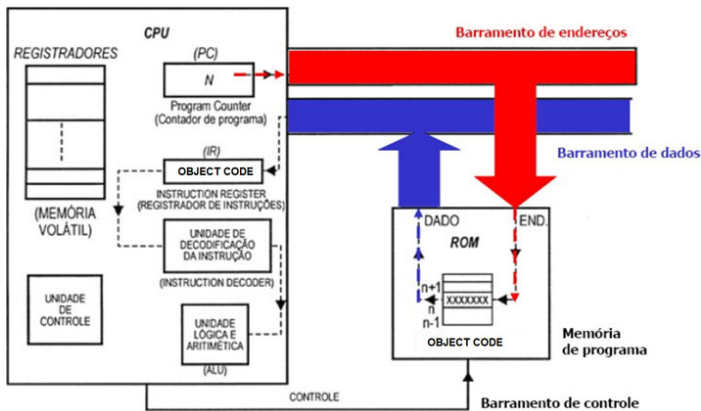
Linguagem de Máquina x Assembly

- Um microcontrolador executa comandos específicos, que são constituídos de números binários;
- Estes comandos ou **object codes** constituem a linguagem de máquina;
- As instruções *assembly* e os comandos de uma linguagem alto nível, são traduzidos em linguagem de máquina.



LOS VERDADEROS PROGRAMADORES
PROGRAMAN EN BINARIO

Linguagem de Máquina

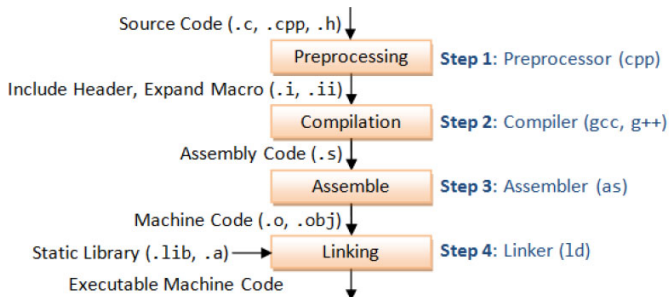


Linguagem Assembly

- Para facilitar a vida do programador, criou-se a Linguagem *Assembly*, que possui o mesmo conjunto de instruções, porém utiliza símbolos (**mnemônicos**) no lugar dos números;
- A conversão da linguagem **assembly** para a linguagem de máquina é feita pelo **assembler** (montador) que, dentre outras coisas, transforma os **mnemônicos** em códigos binários (**opcodes**). Não confunda!
- Entretanto, ainda é específico para cada tipo de CPU, sendo considerada uma linguagem de baixo nível.

Linguagem de Alto Nível

- Há algumas linguagens mais próximas à linguagem humana:
 - C, C++, Pascal, Java, etc.
- De maneira geral, a conversão é feita da seguinte forma:



Linguagem *Assembly ARM Cortex-M4*

Assembly ARM Cortex-M4

- Tecnologia Thumb-2:
 - Mistura de instruções de 16 e 32 bits;
 - Instruções ARM (32) + *Thumb* (16).
- Arquitetura *LOAD/STORE*.
- ▶ Instruções Cortex-M4
- ▶ ARM and Thumb Instructions

Assembly ARM Cortex-M4

- O código fonte do *assembly* é um arquivo de texto (.s ou .asm);
- Por exemplo, uma função que recebe R0 como entrada:

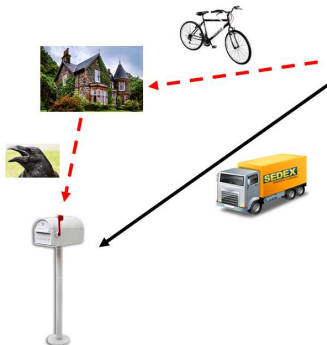
```
1 Func MOV    R1, #100      ; R1=100
2      MUL    R0, R0, R1    ; R0=100*input
3      ADD    R0, #10      ; R0=100*input+10
4      BX     LR           ; retorna 100*input+10
```

Modos de Endereçamento

- As instruções operam com **dados** e **endereços**;
- Formato que a instrução usa para especificar a localização da **memória** para ler ou escrever **dados**.

- Modos:

- Imediato;
- Registrador;
- Indexado;
- Relativo ao PC.



Endereçamento Imediato

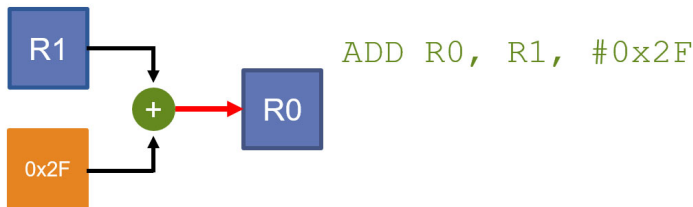
- Uma constante pode ser colocada dentro do código de instrução:
 - Definida por uma **hashtag** ('#') antes do operando;
- Exemplo:



```
1 MOV R0, #25      ;move const. dec 25 para reg R0
2 MOV R1, #0x2F    ;move const. hex 2Fh para reg R1
3 MOV R2, #2_1101  ;move const. bin 00001101 para R2
```

Endereçamento por Registrador

- Algumas instruções podem operar dados com registradores do microprocessador:
- Registrador com modo imediato. Exemplo:



```
1 ADD R1, R2, #18    ;R1 <= R2 + 18
2 AND R0, R1, #0x0F  ;R0 <= R1 & 0x0F
3 MUL R0, R2, #8     ;R0 <= R2 * 8
```

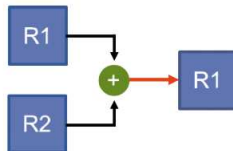
Endereçamento por Registrador

- Exemplos:

MOV R1, R2



ADD R1, R2



```
1 ADD R0, R1, R2 ;R0 <= R1 + R2
2 ADD R3, R4      ;R3 <= R3 + R4
3 MOV R1, R3      ;R1 <= R3
```

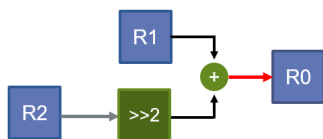
Endereçamento por Registrador

- Exemplo de registrador escalado:

MOV R0, R1, LSL #3



ADD R0, R1, R2, LSR #2



1	MOV	R0, R1, LSL #3	;R0 <= (R1 << 3)
2	ADD	R0, R1, R2, LSR #2	;R0 <= R1 + (R2 >> 2)
3	MOV	R1, R3, ASR #7	;R1 <= R3 / 128 (signed)
4	MOV	R2, R4, LSR #7	;R2 <= R4 / 128 (unsigned)

Endereçamento Indexado

- Instruções ARM não suportam operações de memória para memória (RAM ou ROM);
- Somente as instruções LDR/STR podem acessar a memória;
- Os registradores atuam como ponteiros para a memória:

```
1 LDR R0, [R1]      ;R0 <= [R1]  R0 recebe o dado  
2                  ;apontado por R1  
3 LDR R0, [R1, #0]  ;R0 <= [R1+0] o mesmo que R0 =[R1]  
4 STR R2, [R0, #4]  ;[R0+4] <= R2 guarda o dado R2  
5                  ;endereço apontado por R0 + 4
```

Endereçamento Relativo ao PC

- Se o ponteiro for o *Program Counter* (PC), é usado para:
 - Saltos (*Branches*);
 - Chamadas de funções;
 - Acesso à constantes salvas na ROM.

Endereçamento Relativo ao PC

- Exemplos:

```
1 B      label      ; pula para label
2 BL     subrotina   ; salva PC no R14 (LR) e chama subrotina
3 BX     R14         ; return ou MOV PC, R14 (LR)
4
5 LDR     R1, =Count ; R1 aponta para variável Count
6 LDR     R0, [R1]    ; R0 <= valor apontado por R1
```

Tipos de Operandos

Operandos

- Nas instruções *assembly*, a seguinte lista de símbolos pode ser utilizada:

Instrução	Descrição
Ra Rd Rm Rn Rt e Rt2	Registradores
{Rd,}	Registrador de destino opcional
#imm12	Constante de 12 bits, 0 a 4095
#imm16	Constante de 16 bits, 0 a 65535
operand2	Segundo operando flexível *
{cond}	Condição lógica opcional *
{type}	Estabelece um tipo de dado opcional *
{S}	Opcional: seta os bits de condição do PSR
Rm {, shift}	Deslocamento opcional no Rm
Rn {, offset}	Offset opcional no Rn

- * Descritos a seguir.

Operando2

operand2	
#constant	Valor imediato de 8 bits*
Rm , <opsh>	Registrador, deslocado opcionalmente como abaixo
Rm, LSL Rs	Reg. Rm com desloc. lógico (unsig.) para esquerda definido por Rs
Rm, LSR Rs	Reg. Rm com desloc. lógico (unsig.) para direita definido por Rs
Rm, ASR Rs	Reg. Rm com desloc. aritmético (sig.) para direita definido por Rs
Rm, ROR Rs	Reg. Rm com rotação lógica para direita definido por Rs

- O operando 2 aceita os seguintes valores para a constante:
 - Constante produzida deslocando um valor de 8 bits para esquerda por qualquer número de bits;
 - Constante na forma 0x00XY00XY;
 - Constante na forma 0xXY00XY00;
 - Constante na forma 0xXYXYXYXY.

Tipos de Instruções

Operações de Acesso à Memória

- Acesso à memória de código ou de dados:
 - **LD** lê dados da memória;
 - **ST** escreve dados na memória;
 - Instruções de processamento de dados não acessam a memória;
 - Arquitetura **LOAD-STORE**.
- Para operações com dados em memória, é necessário:
 - Leitura da memória em registrador;
 - Operação;
 - Escrita em memória.

Operações de Acesso à Memória

- Para acessar a memória, **sempre** estabeleça um **registrador ponteiro** (ou base) para o objeto;
- Exemplos:

Estado Inicial

Registradores

R0	0x20000000
R1	0x00
R2	0x00
R3	0x50003210
R4	0x00

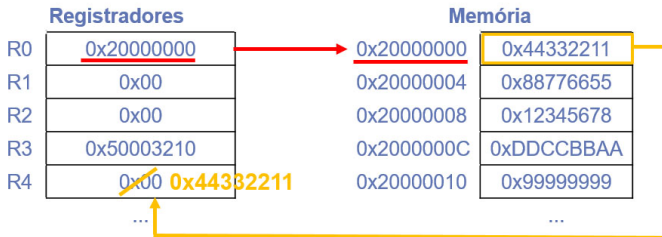
Memória

0x20000000	0x44332211
0x20000004	0x88776655
0x20000008	0x12345678
0x2000000C	0xDDCCBBAA
0x20000010	0x99999999

Operações de Acesso à Memória

- Exemplos:

LDR R4, [R0]



- O primeiro operando é obrigatoriamente por registrador e segundo operando é indexado (`[]`).

Operações de Acesso à Memória

- Exemplos:

STR R3, [R0]



- O primeiro operando é obrigatoriamente por registrador e segundo operando é Indexado (`[]`).

Tipos de Acesso à Memória

- O registrador que contém o endereço ou a localização dos dados é chamado de **registrador base**;
- Utiliza-se o modo de endereçamento indexado;
- Há três tipos: Com *offset*, pré-indexado, pós-indexado.

Tipos de Acesso à Memória

- Com *offset*:

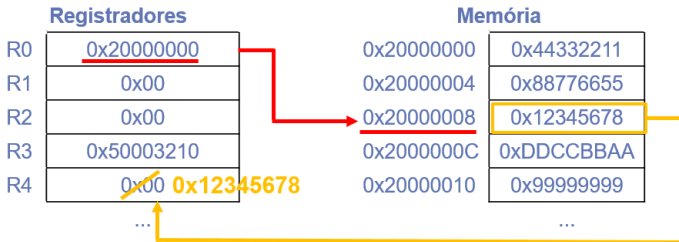
- O endereço é incrementado ANTES da operação, mas o valor do registrador base NÃO é alterado;
- [XX]** → Conteúdo apontado por XX
- Exemplos:

```
1 LDR R0, [R1]           ; R0 <= [R1]  R0 recebe o dado
2                         ; apontado por R1
3 LDR R0, [R1, #8]       ; R0 <= [R1+8] R0 recebe o
4                         ; dado apontado pelo
5                         ; R1 + 8
6 STR R2, [R0, #4]       ; [R0+4] <= R2 guarda o dado
7                         ; R2 ender. apontado por R0+4
```

Acesso à Memória: Com *Offset*

- Exemplo:

LDR R4, [R0, #8]



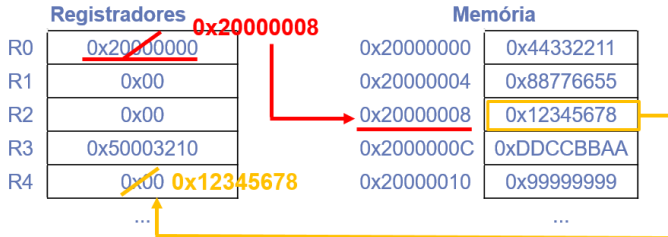
Tipos de Acesso à Memória

- Pré-Indexado → '!':
 - O endereço é incrementado ANTES da operação, e o valor do registrador base é salvo;
- Exemplos:

1	LDR R0, [R1, #2]!	;R0 <= [R1 + 2], R1 <= R1 + 2
2	STR R2, [R3, #4]!	;[R3 + 4] <= R2, R3 = R3 + 4
3	LDR R0, [R1, R2]!	;R0 <= [R1 + R2], R1 <= R1+R2
4	LDR R0, [R1, R2, LSR #2]!	;R0 <= [R1 + (R2 >> 2)]
5		;R1 <= R1 + (R2 >> 2)

Acesso à Memória: Pré-Indexado

- Exemplo:



Tipos de Acesso à Memória

- Pós-Indexado:
 - O endereço é incrementado após a operação e o valor do registrador base é salvo.
- Exemplos:

```
1 LDR    R0, [R1], #8    ;R0 <= [R1]  R1 <= R1 + 8
2 STR    R2, [R3], R4    ;[R3] <= R2, R3 = R3 + R4
3 LDR    R0, [R1], R2, LSR #2 ;R0 <= [R1]
4                                     ;R1 <= R1 + (R2 >> 2)
```

Acesso à Memória: Pós-Indexado

- Exemplo:

LDR R4, [R0], #8



Tipos de Acesso à Memória

- Tabela Comparativa:

Modo	Mnemônico <i>Assembly</i>	Endereço Acessado	Valor Final no Registrador Base
Com Offset, base não alterada	LDR R0, [R1, #d]	$R1 + d$	R1
Pré-indexado, base alterada	LDR R0, [R1, #d]!	$R1 + d$	$R1 + d$
Pós-indexado, base alterada	LDR R0, [R1], #d	R1	$R1 + d$

Operações de Acesso à Memória

- Algumas das instruções *load/store*:

1	LDR {type}{cond}	Rd, [Rn]	<i>;load memory at [Rn] to Rd</i>
2	STR {type}{cond}	Rt, [Rn]	<i>;store Rt to memory at[Rn]</i>
3	LDR {type}{cond}	Rd, [Rn, #n]	<i>;load memory at[Rn+n]to Rd</i>
4	STR {type}{cond}	Rt, [Rn, #n]	<i>;store Rt to memory [Rn+n]</i>
5	LDR {type}{cond}	Rd, [Rn, #n]!	<i>;load memory at[Rn+n]to Rd</i>
6			<i>;Rn := Rn + n</i>
7	STR {type}{cond}	Rt, [Rn, #n]!	<i>;store Rt to memory [Rn+n]</i>
8			<i>;Rn := Rn + n</i>
9	LDR {type}{cond}	Rd, [Rn], #n	<i>;load memory at [Rn] to Rd</i>
10			<i>;Rn := Rn + n</i>
11	STR {type}{cond}	Rt, [Rn], #n	<i>;store Rt to memory [Rn]</i>
12			<i>;Rn := Rn + n</i>

Tipos de dados da memória

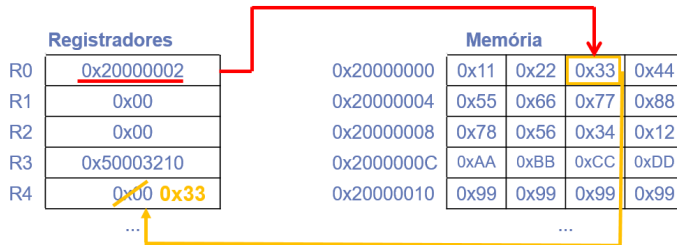
- Em relação a dados de memória, pode-se acessar dados de 8, 16, 32 ou 64 bits. Para 8 e 16 bits pode ser com sinal ou sem sinal;
- Ao colocar um valor de 8 bits ou 16 bits em um registrador:
 - Se for uma operação sem sinal, os bits mais significantes são preenchidos com 0;
 - Se for uma operação com sinal, os bits mais significantes serão iguais ao bit de sinal.
- É **dever** do programador saber como a memória será acessada.

{type}	Tipo do dado	Significado
	Word de 32 bits	0 a +4.294.967.295 ou -2.147.483.648 a +2.147.483.647
B	Byte de 8 bits sem sinal	0 a 255
SB	Byte de 8 bits com sinal	-128 a +127
H	Halfword de 16 bits sem sinal	0 a 65535
SH	Halfword de 16 bits com sinal	-32768 a +32767
D	64-bits	Usa dois registradores

Operações de Acesso à Memória

- Exemplo:

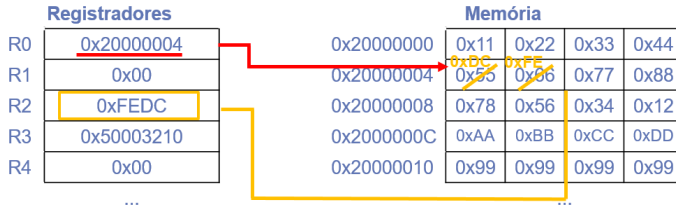
LDRB R4, [R0]



Operações de Acesso à Memória

Exemplo:

STRH R2, [R0]



Operações de Transferência

- Para mover valores entre registradores ou uma constante e um registrador;
- Só conseguimos transferir no máximo valores de até 16 bits:

```
1 MOV{S}{cond} Rd, <op2> ; set Rd equal to the value  
2                           ; specified by op2  
3 MOV{cond} Rd, #im16      ; set Rd equal to im16, im16  
4                           ; is  
5                           ; 0 to 65535  
6 MVN{S}{cond} Rd, <op2> ; set Rd equal to the !value  
                           ; specified by op2
```

- Note que **MVN** inverte os bits do op2 e salva em Rd (equivale ao complemento de 1);
- Já o **NEG** inverte os bits do op2, adiciona 1 e salva em Rd (equivale ao complemento de 2).

Operações de Transferência

- E quando desejamos carregar um registrador com uma constante não suportada pelo **MOV**, o que fazer?

- Usar o **MOV** e **MOVT** (Instrução):

```
1 MOV R1, #0x5678 ;R1[31:16]:=0 R1[15:0]:=0x5678
2 MOVT R1, #0x1234 ;R1[31:16]:=0x1234 R1[15:0]
3 ; não afetada
```

- No Keil (Diretiva):

```
1 LDR R6, Pi ; Definição da constante fora da
2 ; execução do código
3 Pi DCD 314159
```

- Ou:

```
1 LDR R6, =314159
```

- Não confundir com o **LDR** de acesso à memória.

Operações de Transferência

- E quando desejamos carregar um registrador com uma constante não suportada pelo **MOV**, o que fazer?

- Usar o **MOV** e **MOVT** (Instrução):

```
1 MOV R1, #0x5678 ;R1[31:16]:=0 R1[15:0]:=0x5678
2 MOVT R1, #0x1234 ;R1[31:16]:=0x1234 R1[15:0]
3 ; não afetada
```

- No Keil (Diretiva):

```
1 LDR R6, Pi ; Definição da constante fora da
2 ; execução do código
3 Pi DCD 314159
```

- Ou:

```
1 LDR R6, =314159
```

- **Não confundir com o LDR de acesso à memória.**

Exercício: Instruções de Memória/Transfer

- 1 Abra o arquivo no moodle como criar um projeto novo e, em seguida, faça um código que realize os seguintes passos e depois depure no Keil:

(Acrescente ao final do arquivo a instrução **NOP** (do inglês, *No Operation*) para que seja possível depurar o código inteiro)

- a) Salve no registrador R0 o valor 65 decimal;
- b) Salve no registrador R1 o valor 0x1B00.1B00;
- c) Salve no registrador R2 o valor 0x1234.5678;
- d) Guarde na posição de memória 0x2000.0040 o valor de R0;
- e) Guarde na posição de memória 0x2000.0044 o valor de R1;
- f) Guarde na posição de memória 0x2000.0048 o valor de R2;
- g) Guarde na posição de memória 0x2000.004C o número 0xF0001;
- h) Guarde na posição de memória 0x2000.0046 o byte 0xCD, sem sobrescrever os outros bytes da WORD;
- i) Leia o conteúdo da memória cuja posição 0x2000.0040 e guarde no R7;
- j) Leia o conteúdo da memória cuja posição 0x2000.0048 o guarde no R8;
- k) Copie para o R9 o conteúdo de R7.

Operações Lógicas

- Combinar, extrair ou testar uma informação;
- Operações unárias (uma entrada):
 - **Negação;**
 - **Complementar.**
- Operações Binárias (duas entradas):
 - **AND;**
 - **OR;**
 - **XOR.**

Operações Lógicas

- Algumas instruções lógicas:

1	AND	{S}	{ cond }	{ Rd, }	Rn, <op2>	; Rd=Rn&op2
2	ORR	{S}	{ cond }	{ Rd, }	Rn, <op2>	; Rd=Rn op2
3	EOR	{S}	{ cond }	{ Rd, }	Rn, <op2>	; Rd=Rn^op2
4	BIC	{S}	{ cond }	{ Rd, }	Rn, <op2>	; Rd=Rn(~op2)
5	ORN	{S}	{ cond }	{ Rd, }	Rn, <op2>	; Rd=Rn (~op2)

- Adicionar o sufixo 'S' para a condição N ou Z ser atualizada no resultado da operação.

Exercício: Operações Lógicas

- 2) Faça um código que realize os seguintes passos e depois depure no Keil:

(Acrescente ao final do arquivo a instrução **NOP** (do inglês, *No Operation*) para que seja possível depurar o código inteiro)

- a) Realize a operação lógica AND do valor 0xF0 com o valor binário 01010101 e salve o resultado em R0. Utilize o sufixo 'S' para atualizar os bits;
- b) Realize a operação lógica AND do valor 11001100 binário com o valor binário 00110011 e salve o resultado em R1. Utilize o sufixo 'S' para atualizar os bits;
- c) Realize a operação lógica OR do valor 10000000 binário com o valor binário 00110111 e salve o resultado em R2. Utilize o sufixo 'S' para atualizar os bits;
- d) Realize a operação lógica BIC do valor 0xABCDABCD com o valor 0xFFFF0000 e salve o resultado em R3. Utilize o sufixo 'S' para atualizar os bits.

Dúvidas?