

ELF52 - Sistemas Microcontrolados

Linguagem *Assembly - Aritméticas, Condicionais, Pilha, Salto e Diretivas*

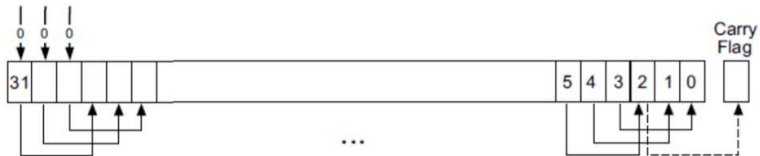
Professor:

Prof. Marcos Eduardo

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

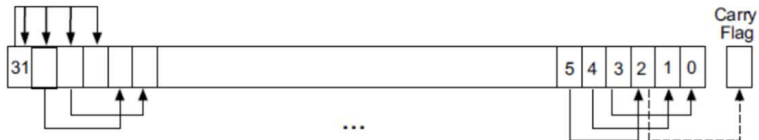
Operações de Deslocamento

- Tem dois parâmetros de entrada e uma saída;
- Deslocamento lógico para direita (LSR{S}):
 - Similar à divisão sem sinal por 2^n ;
 - Um zero é colocado na posição mais significativa.



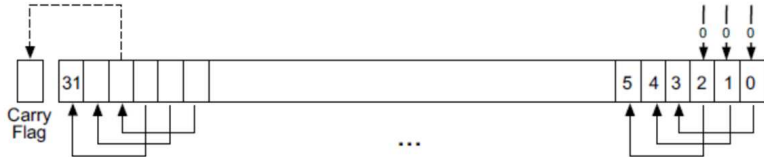
Operações de Deslocamento

- Deslocamento aritmético para direita (ASR{S}):
 - Similar à divisão com sinal por 2^n ;
 - O sinal é preservado.



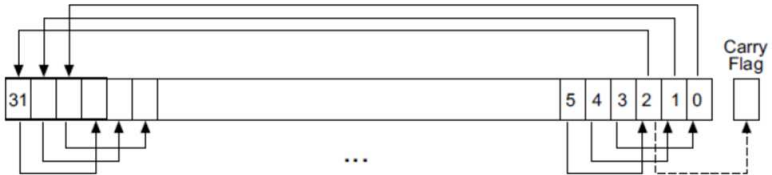
Operações de Deslocamento

- Deslocamento lógico para esquerda (LSL{S}):
 - Similar à multiplicação sem sinal por 2^n ;
 - O sinal não é preservado.



Operações de Deslocamento

- Rotação à direita $\text{ROR}\{S\}$:
 - Gira para a direita o valor dos bits dos registradores;
 - Não há rotação para a esquerda porque uma rotação para a esquerda de n equivale a uma rotação para a direita de $32-n$.



Operações de Deslocamento

- Rotação à direita estendida $RRX\{S\}$:
 - Rotação de UM ÚNICO bit para a direita.



Operações de Deslocamento

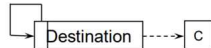
- Resumo:

LSL: Logical Shift Left



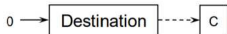
Multiplicação por uma potência de 2

ASR: Arithmetic Right Shift



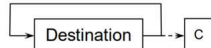
Divisão por uma potência de 2 preservando o sinal

LSR: Logical Shift Right



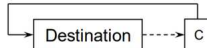
Divisão por uma potência de 2

ROR: Rotate Right



Rotação circular de n bits do LSB para o MSB

RRX: Rotate Right Extended



Rotação circular de 1 bit do flag C para o MSB

Operações de Deslocamento

1	LSR{S}{cond}	Rd, Rm, Rs	; logical shift right Rd=Rm>>Rs
2			; (unsigned)
3	LSR{S}{cond}	Rd, Rm, #n	; logical shift right Rd=Rm>>n
4			; (unsigned)
5	ASR{S}{cond}	Rd, Rm, Rs	; arithmetic shift right
6			; Rd=Rm>>Rs (signed)
7	ASR{S}{cond}	Rd, Rm, #n	; arithmetic shift right
8			; Rd=Rm>>n (signed)
9	LSL{S}{cond}	Rd, Rm, Rs	; shift left Rd=Rm<<Rs (signed,
10			; unsigned)
11	LSL{S}{cond}	Rd, Rm, #n	; shift left Rd=Rm<<n (signed,
12			; unsigned)
13	ROR{S}{cond}	Rd, Rm, Rs	; rotate right
14	ROR{S}{cond}	Rd, Rm, #n	; rotate right
15	RRX{S}{cond}	Rd, Rm	; rotate right 1 bit with extension

Exercício: Operações de Deslocamento

- 3) Faça um código que realize os seguintes passos e depois depure no Keil verificando, na simulação, os valores dos registradores antes e depois:
- a) Realize o deslocamento lógico em 5 bits do número 701 para a direita com o *flag* 'S';
 - b) Realize o deslocamento lógico em 4 bits do número -32067 para a direita com o *flag* 'S' (Use o **MOV** para o número positivo e depois **NEG** para negativar com complemento de 2);
 - c) Realize o deslocamento aritmético em 3 bits do número 701 para a direita com o *flag* 'S';
 - d) Realize o deslocamento aritmético em 5 bits do número -32067 para a direita com o *flag* 'S';
 - e) Realize o deslocamento lógico em 8 bits do número 255 para a esquerda com o *flag* 'S';
 - f) Realize o deslocamento lógico em 18 bits do número -58982 para a esquerda com o *flag* 'S';
 - g) Rotacionar em 10 bits o número 0xFABC1234;
 - h) Rotacionar em 2 bits com o *carry* o número 0x00004321; (Realizar duas vezes).

Operações Aritméticas

- Tipos de operações aritméticas:
 - Soma, subtração, multiplicação, divisão e comparação.
- Executados por meio de hardware digital;
- *Carry*:
 - Soma:
 - 0: soma coube nos 32 bits;
 - 1: soma não coube nos 32 bits.
 - Subtração:
 - 0: se o resultado for negativo;
 - 1: se o resultado for positivo ou zero.
- *Overflow*:
 - Operações com sinal;
 - O bit V é setado quando há uma passagem entre 0x8000.0000 e 0x7FFF.FFFF.

Operações Aritméticas

- Soma e Subtração:

- Nas operações abaixo, quando Rd não é especificado, o resultado é colocado em Rn.

1	ADD {S}{cond} {Rd,} Rn, <op2>	<i>;Rd = Rn + op2</i>
2	ADD {cond} {Rd,} Rn, #im12	<i>;Rd = Rn + im12</i>
3	ADC {S}{cond} {Rd,} Rn, <op2>	<i>;Rd = Rn + op2 + C</i>
4	SUB {S}{cond} {Rd,} Rn, <op2>	<i>;Rd = Rn - op2</i>
5	SUB {cond} {Rd,} Rn, #im12	<i>;Rd = Rn - im12</i>
6	RSB {S}{cond} {Rd,} Rn, <op2>	<i>;Rd = op2 - Rn</i>
7	RSB {cond} {Rd,} Rn, #im12	<i>;Rd = im12 - Rn</i>
8	CMP {cond} Rn, <op2>	<i>;Rn - op2</i>
9	CMN {cond} Rn, <op2>	<i>;Rn - (-op2)</i>

- CMP** e **CMN** apenas criam condições de comparação para *if-then* e *loops*.

Operações Aritméticas

- Multiplicação e divisão (Resultado em 32 bits):
 - Nas operações abaixo, quando Rd não é especificado, o resultado é colocado em Rn.

1	MUL {S}{cond} {Rd,} Rn, Rm	$;Rd = Rn * Rm$
2	MLA {cond} Rd, Rn, Rm, Ra	$;Rd = Ra + Rn * Rm$
3	MLS {cond} Rd, Rn, Rm, Ra	$;Rd = Ra - Rn * Rm$
4	UDIV {cond} {Rd,} Rn, Rm	$;Rd = Rn / Rm \text{ unsigned}$
5	SDIV {cond} {Rd,} Rn, Rm	$;Rd = Rn / Rm \text{ signed}$

- Multiplicação (Resultado em 64 bits):

1	UMULL {cond} RdLo, RdHi, Rn, Rm	$;Rd = Rn * Rm$
2	SMULL {cond} RdLo, RdHi, Rn, Rm	$;Rd = Rn * Rm$
3	UMLAL {cond} RdLo, RdHi, Rn, Rm	$;Rd = Rd + Rn * Rm$
4	SMLAL {cond} RdLo, RdHi, Rn, Rm	$;Rd = Rd + Rn * Rm$

Exercício: Operações Aritméticas

- 4 Faça um código que realize os seguintes passos e depois depure no Keil verificando, na simulação, os valores dos registradores antes e depois:

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

- a) Adicione os números 101 e 253 atualizando os *flags*;
- b) Adicione os números 1500 e 40543 sem atualizar os *flags*;
- c) Subtraia o número 340 pelo número 123 atualizando os *flags*;
- d) Subtraia o número 1000 pelo número 2000 atualizando os *flags*;
- e) Multiplique o número 54378 por 4; (Essa operação é semelhante a qual?)
- f) Multiplique com o resultado em 64 bits os números 0x11223344 e 0x44332211
- g) Divida o número 0xFFFF7560 por 1000 com sinal;
- h) Divida o número 0xFFFF7560 por 1000 sem sinal;

Bloco If-Then

- Um bloco IT consiste de uma a quatro instruções condicionais, as condições devem ser coerentes;
- Sintaxe:

```
1 IT {x{y{z}}} {cond}
```

- $x \rightarrow$ especifica a condição para a segunda instrução no bloco (T: executa se verdadeiro; E: executa caso contrário);
- $y \rightarrow$ especifica a condição para a terceira instrução no bloco (T: executa se verdadeiro; E: executa caso contrário);
- $z \rightarrow$ especifica a condição para a quarta instrução no bloco (T: executa se verdadeiro; E: executa caso contrário).

- Exemplo:

```
1 ITTE NE           ; começo do bloco
2   ANDNE   R0,R0,R1
3   ADDSNE  R2,R2,#1
4   MOVEQ   R2,R3
```

Bloco *If-Then*

Note

Your assembler might be able to generate the required IT instructions for conditional instructions automatically, so that you do not need to write them yourself. See your assembler documentation for details.

Bloco *If-Then*

Note

Your assembler might be able to generate the required IT instructions for conditional instructions automatically, so that you do not need to write them yourself. See your assembler documentation for details.



Códigos de Condição

Sufixo	Descrição	Flags	Sufixo	Descrição	Flags
EQ	<i>Equal</i>	Z=1	HI	<i>Higher, unsigned</i>	C=1 && Z=0
NE	<i>Not Equal</i>	Z=0	LS	<i>Lower or same, unsigned</i>	C=0 Z=1
CS ou HS	<i>Higher or same, unsigned</i>	C=1	GE	<i>Greater than or equal, signed</i>	N = V
CC ou LO	<i>Lower, unsigned</i>	C=0	LT	<i>Less than, signed</i>	N!=V
MI	<i>Negative</i>	N=1	GT	<i>Greater than, signed</i>	Z=0 && N = V
PL	<i>Positive or zero</i>	N=0	LE	<i>Less than or equal, signed</i>	Z=1 && N!=V
VS	<i>Overflow</i>	V=1	AL	<i>Always. Default</i>	
VC	<i>No overflow</i>	V=0			



► Mais informações

Bloco *If-Then*

- A instrução de salto condicional $B\{\text{cond}\}$ label é a única que **não precisa** estar em bloco IT;
- As instruções IT, CBZ, CBNZ, CPSIE, CPSID **não podem** estar em bloco IT;
- Uma instrução que altera o PC, só pode estar em um bloco IT se for a última instrução do bloco.

Exercício: Bloco *If-Then*

- 5) Faça um código que realize os seguintes passos e depois depure no Keil verificando, na simulação, os valores dos registradores antes e depois:

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

- a) Mova o valor 10 para o registrador R0;
- b) Teste se o registrador é maior ou igual que 9;
- c) Crie um bloco com *If-Then* com 3 execuções condicionais:
 - 1 Se sim, salve o número 50 no R1;
 - 2 Se sim, adicione 32 com o R1 e salve o resultado em R2;
 - 3 Se não, salve o número 75 no R3.
- d) Agora verifique se o registrador é maior ou igual a 11 e execute novamente o passo (c)

Pilha (*Stack*)

- Região da RAM para armazenamento temporário;
- *Last-in-First-out* (LIFO);
- Opera sempre em 32 bits;
- Para salvar um registrador na pilha **PUSH**;
- Para restaurar um registrador da pilha **POP**;
- Por padrão, SP (R13) aponta para o topo da pilha e é decrementado de 4 bytes a cada **PUSH** e incrementado de 4 bytes a cada **POP** (*Full Descending Stack*).

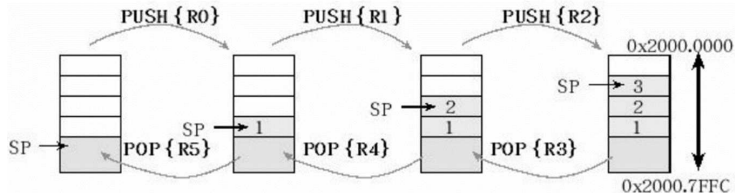
Pilha (Stack)

- Comandos:

```
1 PUSH <reglist>
2 POP <reglist>
```

- <reglist> → Lista de registradores separada entre ',' entre chaves.
Ex: {R0, R1, R4-R9};

- Exemplo de operação:



Exercício: Pilha (*Stack*)

- 6 Faça um código que realize os seguintes passos e depois depure no Keil verificando, na simulação, os valores dos registradores antes e depois:

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

- a) Mova o valor 10 para o registrador R0;
- b) Mova o valor 0xFF11CC22 para o registrador R1;
- c) Mova o valor 1234 para o registrador R2;
- d) Mova o valor 0x300 para o registrador R3;
- e) Empurre para a pilha o R0;
- f) Empurre para a pilha os R1, R2 e R3;
- g) Visualize a pilha na memória (o topo da pilha está em 0x2000.0400);
- h) Mova o valor 60 para o registrador R1;
- i) Mova o valor 0x1234 para o registrador R2;
- j) Desempilhe corretamente os valores para os registradores R0, R1, R2 e R3.

Instruções de Salto

- Instruções para interromper o fluxo ou chamar subrotinas:

```

1 B{cond} label    ;branch to label
2 BX{cond} Rm      ;branch indirect to location
                    specified by Rm
3 BL{cond} label   ;branch to subroutine at label

```

- Instrução B*{cond} (incluindo CBZ e CBNZ) é a única que não precisa estar dentro de um bloco *if-then*;
- Outras instruções de salto **condicional** (não muda *flags* de condição, compara com zero e só pode pular **para frente** de 4 a 130 bytes):

```

1 CBZ    Rn, <label>    ;compare and branch if zero
2 CBNZ   Rn, <label>    ;compare and branch if non
                    zero

```

Exercício: Instruções de Salto

- 7) Faça um código que realize os seguintes passos e depois depure no Keil verificando, na simulação, os valores dos registradores antes e depois:
- a) Mover para o R0 o valor 10;
 - b) Somar R0 com 5 e colocar o resultado em R0;
 - c) Enquanto a resposta não for 50 somar mais 5;
 - d) Quando a resposta for 50 chamar uma função que:
 - 1 Copia o R0 para R1;
 - 2 Verifica se R1 é menor que 50;
 - 3 Se for menor que 50 incrementa, caso contrário modifica para -50.
 - e) Depois que retornar da função coloque uma instrução NOP;
 - f) Acrescente uma instrução para ficar travado na última linha de execução.

Diretivas do *Assembler* do Keil

- Auxiliam o processo de montagem (*assembly*);
- Não fazem parte do conjunto de instruções:
 - **CODE**: espaço para instruções de máquina (ROM);
 - **DATA**: espaço para variáveis globais (RAM);
 - **STACK**: espaço para pilha (RAM);
 - **ALIGN=n**: começa a área alinhada para 2^n bytes;
 - **|.text|**: seções de código produzidas pelo compilador C. Faz o código *assembly* poder ser chamado do C;
 - **NOINIT**: faz uma área da RAM ser não inicializada.

Diretivas do *Assembler* do Keil

```

1 AREA RESET,CODE,READONLY      ;reset vectors in flash ROM
2 AREA DATA                     ;places objects in data memory
   (RAM)
3 AREA |.text|,CODE,READONLY,ALIGN=2      ;code in
   flash ROM
4 AREA STACK,NOINIT,READWRITE,ALIGN=3    ;stack area

```

- Para linkar variáveis e funções entre dois arquivos:

- **EXPORT** ou **GLOBAL**: no arquivo onde o objeto está definido;
- **IMPORT** ou **EXTERN**: no arquivo que está tentando acessar.

```

1 IMPORT name                    ;imports function "name" from
   other file
2 EXPORT name                    ;exports public function "name"
   for use
3                               ;elsewhere

```

Diretivas do *Assembler* do Keil

- **ALIGN**: garante que o próximo objeto será alinhado propriamente. Recomendável colocar ao final do arquivo:

```
1 ALIGN      ;skips 0 to 3 bytes to make next word  
    aligned  
2 ALIGN 2    ;skips 0 or 1 byte to make next halfword  
    aligned  
3 ALIGN 4    ;skips 0 to 3 bytes to make next word  
    aligned
```

- **THUMB**: colocada no topo do arquivo para especificar que o código será gerado com instruções Thumb:

```
1 THUMB
```

- **END**: Diretiva no fim de cada arquivo.

Diretivas do *Assembler* do Keil

- Adicionando variáveis e constantes:

```

1 DCB  expr{,expr}           ;places 8-bit byte(s) into
   memory
2 DCW  expr{,expr}           ;places 16-bit halfword(s)
   into memory
3 DCD  expr{,expr}           ;places 32-bit word(s) into
   memory
4 SPACE size                  ;reserves size bytes,
   uninitialized
  
```

- Como fazer um vetor de 5 bytes?

```

1 ;Declarar na região da RAM (abaixo do AREA DATA)
2 nome_do_vetor    SPACE    5    ;(5 bytes)
3
4 ;Na região do código quando for utilizar
5 LDR  R0, =nome_do_vetor      ;(região inicial)
6 LDRB R1, [R0]                ;(vetor de bytes)
  
```

Diretivas do *Assembler* do Keil

- DCB, DCW e DCD podem ser colocados na área de dados (alocando um espaço no qual o nome da variável retorna o endereço constante dele):

```

1 ;Declarar na região da RAM (abaixo do AREA DATA)
2 variavel DCB 0x00 ; cria a variável (0x00 ignorado)
3 ;Na região do código quando for utilizar
4 LDR R1, =variavel ;pega o endereço na RAM
5 STR R2, [R1] ;salva R2 na variável

```

- Ou na memória de programa (criando constantes):

```

1 ;Na região do código quando for utilizar
2 LDR R5, =STR_1 ;pega o end. do prim. elemento
3 LDRB R6, [R5], #1 ;pega o valor do prim. elemento
4 ;Após o código, para não desalinhar
5 STR_1 DCB "UTFPR",0
6 VETOR_1 DCB 0xFF, 0xAA, 0x12, 0x14,0

```

Diretivas do *Assembler* do Keil

- **EQU**: define um nome simbólico à uma constante numérica:

```
1 GPIO_PORTD_DATA_R EQU 0x400073FC  
2 GPIO_PORTD_DIR_R  EQU 0x40007400  
3 GPIO_PORTD_DEN_R  EQU 0x4000751C
```

Exercício: Diretivas do *Assembler* do Keil

- 8) Faça um código que realize os seguintes passos e depois depure no Keil verificando, na simulação, os valores dos registradores antes e depois:
 - a) Ler de uma posição da memória RAM um número e calcular o fatorial. Armazenar o resultado em R0.

Dúvidas?