175480_03

Patrick de Carvalho Tavares Rezende Ferreira

EA871

Atividade 1:

1- As instruções THUMB fazem parte de um conjunto de instruções ARM que são codificadas em 16 bits, porém possuem formas correspondentes às instruções de 32 bits. Assim há uma economia de espaço na memória.

2- A diretiva ".data" indica que a declaração seguinte será alocada na área da memória destinada aos dados.

3- A diretiva do montador ".text" indica que as declarações seguintes serão alocadas na área da memória onde ficam armazenadas as instruções do código.

4- A diretiva ".align n" alinha os dados em endereços da memória múltiplos de endereço múltiplo de (2^n).

5-Inicializa uma parte da memória com uma constante de 32 Bits.

Atividade 2:

1-

Tabela Disassembler

Endereço	Conteúdo	Rótulo	Mnemônico Instrução
0x00000800	0x4b11	main:	ldr r3,SIM_SCGC5
0x00000802	0x4A11		ldr r2,SIM_SCGC5
0x00000804	0x6812		ldr r2, [r2, #0]
0x00000806	0x2180		movs r1, #128
0x00000808	0x00C9		lsl r1, r1, #3
0x0000080A	0x430A		orr r2, r1
0x0000080C	0x601A		str r2, [r3, #0]
0x0000080E	0x4B0F		ldr r3, PORTB_PCR18
0x00000810	0x2280		movs r2, #128
0x00000812	0x0052		lsl r2, r2, #1
0x00000814	0x601A		str r2, [r3, #0]

0x00000816	0x4B0E		ldr r3, GPIOB_PDDR
0x00000818	0x4A0D		ldr r2, GPIOB_PDDR
0x0000081A	0x6812		ldr r2, [r2, #0]
0x0000081C	0x2180		movs r1, #128
0x0000081E	0x02C9		lsl r1, r1, #11
0x00000820	0x430A		orr r2, r1
0x00000822	0x601A		str r2, [r3, #0]
0x00000824	0x4B0B	loop:	ldr r3, GPIOB_PTOR
0x00000826	0x2280		movs r2, #128
0x00000828	0x02D2		lsl r2, r2, #11
0x0000082A	0x601A		str r2, [r3, #0]
0x0000082C	0x4B0B	espera:	ldr r3, VAR_TEMPO
0x0000082E	0x4A0A		ldr r2, TEMPO
0x00000830	0x601A		str r2, [r3, #0]
0x00000832	0xE004		b testa
0x00000834	0x4B09	decrementa:	ldr r3, VAR_TEMPO
0x00000836	0x681B		ldr r3, [r3, #0]
0x00000838	0x1E5A		sub r2, r3, #1
0x0000083A	0x4B08		ldr r3, VAR_TEMPO
0x0000083C	0x601A		str r2, [r3, #0]
0x0000083E	0x4B07	testa:	ldr r3, VAR_TEMPO
0x00000840	0x681B		ldr r3, [r3, #0]
0x00000842	0x2B00		cmp r3, #0
0x00000844	0xD1F6		bne decrementa
0x00000846	0xE7ED		b loop

0x00000848	0x40048038	SIM_SCGC5:	
0x0000084C	0x4004A048	PORTB_PCR18:	
0x00000850	0x400FF054	GPIOB_PDDR:	
0x00000854	0x400FF04C	GPIOB_PTOR:	
0x00000858	0x0007A120	TEMPO:	
0x0000085C	0x1FFFF000	VAR_TEMPO:	

Tabela de Símbolos

Símbolo	Valor
main	0x00000800
loop	0x00000824
espera	0x0000082C
decrementa	0x00000834
testa	0x0000083E
SIM_SCGC5	0x00000848
PORTB_PCR18	0x0000084A
GPIOB_PDDR	0x0000084C
GPIOB_PTOR	0x0000084E
TEMPO	0x00000850
VAR_TEMPO	0x00000852
RAM_START	0x1FFFF000

2-

Primeira instrução: “ldr r3,SIM_SCGC5”

Obtém o valor salvo no endereço de SIM_SCGC5 e o copia para o registrador r3.

Possui a seguinte codificação:

Bits	15 - 11	10 – 8	7 – 0
Finalidade	Código da Operação (01001)	Registrador Destino	Offset

Segunda instrução: “ldr r2,SIM_SCGC5”

Obtém o valor salvo no endereço de SIM_SCGC5 e o copia para o registrador r2.

Possui a seguinte codificação:

Bits	15 - 11	10 – 8	7 – 0
Finalidade	Código da Operação (01001)	Registrador Destino	Offset

Terceira instrução: “ldr r2, [r2, #0]”

Obtém o valor salvo no endereço de r2, faz o pré-incremento de valor 0 no conteúdo de r2 (onse está o endereço base) e o copia para o registrador r2.

Possui a seguinte codificação:

Bits	15 - 11	10 – 8	7 – 0
Finalidade	Código da Operação (01001)	Registrador Destino	Endereço base + incremento (0)

Quarta instrução: “movs r1, #128”

Esta instrução possui a seguinte forma: movs <Rd>, #<const>, onde Rd é registrador destino (r1) que recebe const (valor imediato, no caso, 128).

Possui a seguinte codificação:

Bits	15 - 11	10 – 8	7 – 0
Finalidade	Código da Operação (00100)	Registrador Destino	Imediato (128)

Quinta instrução: “lsl r1, r1, #3”

Sob a forma “lsl <Rd>, <Rm>, #<imm5>” , a instrução lsl faz um shift lógico no valor de Rm por um número de casas salvo em imm5, guardando o resultado em Rd.

Possui a seguinte codificação:

Bits	15 - 11	10 – 6	5 – 3	2 - 0
Finalidade	Código da Operação (00000)	Imediato (3)	Registrador (r1)	Registrador Destino (r1)

Sexta instrução: “orr r2, r1”

“orr <Rd>, <Rm>” realiza a operação OU lógica bit-a-bit entre os valores binários de Rd e Rm salvando o resultado em Rd.

Possui a seguinte codificação:

Bits	**15 – 6**	**5 – 3**	**2 - 0**
Finalidade	Código da Operação (0100001100)	Registrador (r1)	Registrador Destino (r2)

Sétima instrução: “str r2, [r3, #0]”

“str <Rt>, [Rm, #<imm8>]” Armazena o conteúdo de Rm (r3) em Rd(r2), pós-incrementando Rm no valor de imm8.

Possui a seguinte codificação:

Bits	**15 –11**	**10 – 8**	**7 - 0**
Finalidade	Código da Operação (10010)	Registrador (r2)	Registrador Destino + Imediato (r2)

Oitava instrução: “ldr r3, PORTB_PCR18”

Obtém o valor salvo no endereço de PORTB_PCR18 e o copia para o registrador r3.

Possui a seguinte codificação:

Bits	**15 - 11**	**10 – 8**	**7 – 0**
Finalidade	Código da Operação (01001)	Registrador Destino	Offset

Nona instrução: “movs r2, #128”

Esta instrução possui a seguinte forma: movs <Rd>, #<const>, onde Rd é registrador destino (r2) que recebe const (valor imediato, no caso, 128).

Possui a seguinte codificação:

Bits	**15 - 11**	**10 – 8**	**7 – 0**
Finalidade	Código da Operação (00100)	Registrador Destino	Imediato (128)

Décima instrução: “lsl r2, r2, #1”

Sob a forma “lsl <Rd>, <Rm>, #<imm5>” , a instrução lsl faz um shift lógico no valor de Rm por um número de casas salvo em imm5, guardando o resultado em Rd.

Possui a seguinte codificação:

Bits	**15 - 11**	**10 – 6**	**5 – 3**	**2 - 0**
Finalidade	Código da Operação (00000)	Imediato (1)	Registrador (r2)	Registrador Destino (r2)

3-

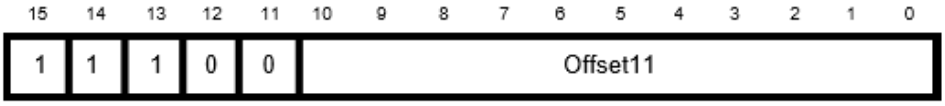
Ao remover a instrução align 2, o funcionamento do código permanece inalterado, porque sem esta diretiva, o programa executa instruções de 16bits e, com a diretiva, executa instruções de 32 bits.

Após a aplicação da instrução nop, o programa não executa corretamente, pois as palavras da memória não estão alinhadas. Dessa forma, quando utilizamos inserimos a diretiva align, o programa volta à sua execução normal, pois agora as palavras na memória estão alinhadas.

4-

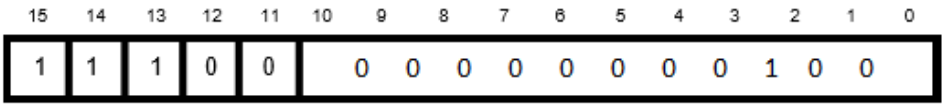
As instruções b<label> fazem saltos incondicionais para o endereço de label e possuem a seguinte codificação:

Figura: Codificação binária instrução branch (b).



Para se calcular offset de b testa, somamos o endereço desta instrução ao incremento feito pelo pc (0x832 + 0x4 = 0x836). Em seguida, verificamos o endereço de teste (0x83E), para calcular o offset, que é a "distância" entre os dois endereços (A diferença entre estes números). Logo, 0x83E-0x836=0x8=0b1000. Sabe-se que o deslocamento será par, logo, não é necessário codificar o bit menos significativo. Assim, a codificação da instrução fica da seguinte forma:

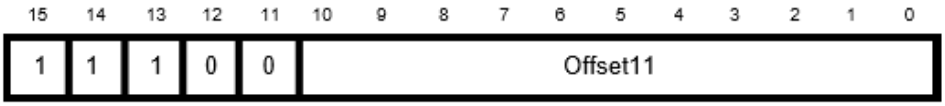
Figura: Codificação binária instrução branch testa (b testa).



5-

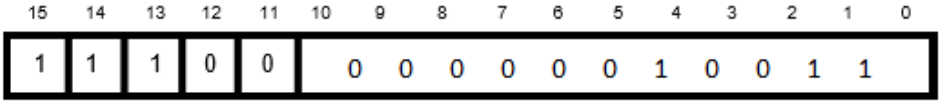
As instruções b<label> fazem saltos incondicionais para o endereço de label e possuem a seguinte codificação:

Figura: Codificação binária instrução branch (b).



Para se calcular offset de b loop, somamos o endereço desta instrução ao incremento feito pelo pc (0x846 + 0x4 = 0x84A). Em seguida, verificamos o endereço de loop (0x824), para calcular o offset, que é a "distância" entre os dois endereços (A diferença entre estes números). Logo, 0x84A-0x824=0x26=0b100110. Sabe-se que o deslocamento será par, logo, não é necessário codificar o bit menos significativo. Assim, a codificação da instrução fica da seguinte forma:

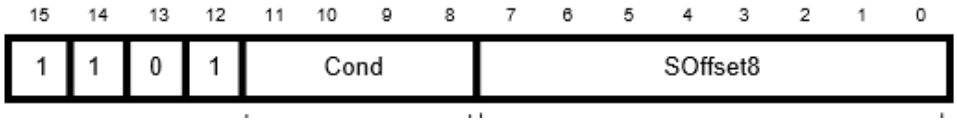
Figura: Codificação binária instrução branch loop (b loop).



6-

bne decrementa é é um branch para o rótulo decrementa. Porém, este só é efetuado se a flag de "not equal" estiver ativa. Logo, "ne" é um condicinal para a instrução de branch.

Figura: Codificação binária instrução branch condicional(b<cond>).



Para se calcular offset de bne decrementa, somamos o endereço desta instrução ao incremento feito pelo pc (0x844 + 0x4 = 0x848). Em seguida, verificamos o endereço de decrementa(0x834), para calcular o offset, que é a "distância" entre os dois endereços (A diferença entre estes números). Logo, 0x834-0x848=0x-14=0b11111110010(em complemento de 2).

Temos que o código para o condicional de "not equal" é 0001 e sabe-se que o deslocamento será par, logo, não é necessário codificar o bit menos significativo. Assim, a codificação da instrução fica da seguinte forma:

Figura: Codificação binária instrução branch condicional(bne decrementa).