Graded ARM assembly language Examples

These examples have been created to help students with the basics of Keil's ARM development system. I am providing a series of examples that demonstrate the ARM's instruction set.

These begin with very basic examples of addition. If any reader has difficulties with this material or can suggest improvements or corrections, please email me at <u>alanclements@ntlworld.com</u> and I will do my best to make the appropriate changes.

In most examples, I present the problem in words, the basic code, and then the assembly language version. I also show the output of the simulator at various stages during the simulation. When writing assembly language I use bold font to indicate the destination operand (this will be in the code fragments as the assembler does not support this).

Quick Guide to Using the Keil ARM Simulator

- 1. Run the IDE package. I am using μ Vision V4.22.22.0
- 2. Click **Project**, select **New µMicrovision Project** Note that bold blue font indicates your input to the computer and bold blue indicates the computer's response (or option).
- 3. Enter filename in the **File name** box. Say, **MyFirstExample**
- 4. Click on Save.
- 5. This causes a box labelled **Select Device for Target 'Target 1'** to pop up. You now have to say which processor family and which version you are going to use.
- 6. From the list of devices select ARM and then from the new list select ARM7 (Big Endian)
- 7. Click on OK. The box disappears. You are returned to the main µVision window.
- 8. We need to enter the source program. Click **File**. Select **New** and click it. This brings up an edit box labelled **Text1**. We can now enter a simple program. I suggest:

```
AREA MyFirstExample, CODE, READONLY
ENTRY
MOV r0,#4 ;load 4 into r0
MOV r1,#5 ;load 5 into r1
ADD r2,r0,e1 ;add r0 to r1 and put the result in r2
S B S ;force infinite loop by branching to this line
END ;end of program
```

- 9. When you've entered the program select **File** then **Save** from the menu. This prompts you for a **File name**. Use **MyFirstExample.s** The suffix **.s** indicates source code.
- 10. This returns you to the window now called **MyFirstExample** and with the code set out using ARM's conventions to highlight code, numbers, and comments.
- 11. We now have to set up the environment. Click **Project** in the main menu. From the pulldown list select **Manage**. That will give you a new list. **Select Components, Environment, Books..**
- 12. You now get a form with three windows. Below the right hand window, select Add Files.
- 13. This gives you the normal Windows file view. Click on the File of type expansion arrow and select Asm Source file (*.s*; *.src; *.a*). You should now see your own file MyFirstExample.s appear in the window. Select this and click the Add tab. This adds your source file to the project. Then click Close. You will now see your file in the rightmost window. Click OK to exit.
- 14. That's it. You are ready to assemble the file.
- 15. Select **Project** from the top line and then click on **Built target**.
- 16. In the bottom window labelled **Build Output** you will see the result of the assembly process.
- 17. You should see something like:

Build target 'Target 1' assembling MyFirstExample.s... linking... Program Size: Code=16 RO-data=0 RW-data=0 ZI-data=0 ''MyFirstExample.axf'' - 0 Error(s), 0 Warning(s).

18. The magic phrase is **"0 Error(s)"**. If you don't get this you have to re-edit the source file. And then go to **Project** and **Build target** again.

Example 1 ADDITION

The problem: P = Q + R + SLet Q = 2, R = 4, S = 5. Assume that r1 = Q, r2 = R, r3 = S. The result Q will go in r0.

The Code ADD r0,r1,r2 ;add Q to R and put in P ADD r0,r0,r3 ;add S to P and put the result in P

The program

```
AREA Example1, CODE, READONLY
ADD r0,r1,r2
ADD r0,r3
Stop B Stop
END
```

Notes:

- 1. The semicolon indicates a user-supplied comment. Anything following a semicolon on the same line is ignored by the assembler.
- 2. The first line is AREA Example1, CODE, READONLY is an assembler directive and is required to set up the program. It is a feature of the development system and not the ARM assembly language. An assembler from a different company may have a different way of defining the start of a program. In this case, AREA refers to the segment of code, Example1 is the name we've given it, CODE indicates executable code rather than data, and READONLY state that it cannot be modified at run time.
- 3. Anything starting in column 1 (in this case Stop) is a label that can be used to refer to that line.
- 4. The instruction Stop B Stop means 'Branch to the line labelled Stop' and is used to create an infinite loop. This is a convenient way of ending programs in simple examples like these.
- 5. The last line END is an assemble directive that tells the assembler there is not more code to follow. It ends the program.

Figure Example 1.1 shows the screed after the program has been loaded and the simulation mode entered. You can configure this window. We have included the disassembly window that shows the code in memory and converts it into instructions. This feature can be confusing because it will take data and try and convert it into instructions. However, the feature is useful because it shows the actual ARM instructions that are created by your instructions. We will see that some instructions that you write are pseudo instructions that are translated into appropriate ARM instructions.

Figure Example 1.1 The state of the system after loading the code for Example 1

🔣 E:\CengageEditi	on_2\StructuredARM	lexamples\Example1_ADDITION.uvproj - µVision4	
<u>F</u> ile <u>E</u> dit <u>V</u> iew	<u>P</u> roject Fl <u>a</u> sh <u>D</u> eb	bug Pe <u>r</u> ipherals <u>T</u> ools <u>S</u> VCS <u>W</u> indow <u>H</u> elp	
i 🗋 💕 🛃 🧊	※ 単語 タや	😔 塑整整数 導 導 /// 振 🙋 📃 🔍 🔍 🥥	• • ठ
🚼 🖾 🖓	} () +() +() ↓		
Registers	μ×	Disassembly	д×
Register	Value	3: ADD r0,r1,r2 ;P = Q + R	
	_	C>0x0000000 E0810002 ADD R0,R1,R2	
RO	0x0000000	4: ADD r0,r3 ;P = P + S	
R1	0x0000000	0x00000004 E0800003 ADD R0,R0,R3	
R2	0x0000000	S: Stop B Stop ; rail through to an infinite for	,p
R3	0x0000000	0x0000000C 00000000 ANDEO 80.80.80	
R4	0x0000000	0x00000010 00000000 ANDEO R0.R0.R0	
R5	0x0000000	0x00000014 00000000 ANDEQ R0,R0,R0	
D7	0x0000000	0x00000018 00000000 ANDEQ R0,R0,R0	_
B8	0x0000000		
R9	0x0000000	Example1 ADDITION s	▼ X
R10	0x0000000		
R11	0x0000000	AREA Example1, CODE, READONLY	A
R12	0x0000000	$\frac{2}{1+2}$	
R13 (SP)	0x0000000	4 ADD r_0, r_3 $P = P + S$	
R14 (LR)	0x0000000	5 Stop B Stop ;Fall through to an infinite loop	
R15 (PC)	0x0000000	6	
	0x00000003	7 END ;This ends the program	
H. User/System		8	
± Fast Interrupt			
± Interrupt			
• Supervisor			
+ Abort			
🗄 🖽 Undefined			
⊡ Internal			
PC \$	0x0000000		
Mode	Supervisor		
States	0 0000000		
Jec	0.000000		
🖭 Project 🛛 🚟 Reg	isters		
			Sim

Because there is no means of entering initial data into registers, you have to do that manually. Simply double-click a register and then modify its value.

Figure Examp	le 1.2 The state	of the system after	funning the code.
--------------	------------------	---------------------	-------------------

💘 E:	\Cenga	geEditi	on_2\Str	ucture	dARMe	exampl	es\Exam	ple1_A	DDITION	l.uvproj - j	uVision4		
<u>F</u> ile	<u>E</u> dit	<u>V</u> iew	<u>P</u> roject	Fl <u>a</u> sh	<u>D</u> eb	ug Pej	ripherals	<u>T</u> ools	<u>s</u> vcs	<u>W</u> indow	<u>H</u> elp		
	🚰		3 Pa 6	1 S	6		PB	四日	N 💷	軍 //E //j	į 🖄		
RST		3 P	} {} {}	*{} 4			3 📰 🖗	5 👼 •	- 💷 - [🛃 • 🔜 ·	- 💷 -	👿 - 🔀	- 3
Regis	ters				μ Χ	Disass	embly						μ×
Regi	ister		Value				2:		ADD	r0,r1,	r2		_
	Current					0x0	000000)0 E	081000	2 ADD		R0,R1,R	2 —
	R0		0x0000	000B			3:		ADD	r0,r3			
	R1		0x0000	0002		0x0	000000)4 E	080000	3 ADD		RO,RO,R	3
	R2		0x0000	0004			4: 5	Stop	В	Stop			
	R3		0x0000	0005			000000)8 E	AFFFFFI	EВ	_	0x00000	800
	R4		0x0000	0000		0x0	000000	oc o	000000	0 ANDE	2	RO,RO,RO	D
	R5		0x0000	0000		0x0	000001	0 0	000000	0 ANDE	2	RO,RO,R	D
	R6		0x0000	0000		0x0	000001	.4 0	000000	0 ANDE	2	RO,RO,RO	
	R7		0x0000	0000		0x0	000001	.8 0	000000	0 ANDE	Q	RO,RO,RO	
	R8		0x0000	0000									
	R9		0x0000	0000			Examp	le1 AD	DITION.s				▼ ×
	R10		0x0000	0000					Examp				
	R11		0x0000	0000				ADD	r0.r1	.r2	DE, K	EADONET	
	R12		0x0000	0000				ADD	r0,r3				
	R13	(SP)	0x0000	0000			Stop	В	Stop				
	R14	(LR)	0x0000	0000	-			END					-
🖭 Pi	roject	📰 Reg	isters										

Note that the contents of r0 are 2 + 4 + 5 = 11 = 0x0B. This is the result we expected.

Running a Program in the Simulator

Having loaded a program as code and assembled it, you need to run it. You can either continue from where you left off after assembling the code with **Build target**, or you can start afresh and load the code.

If you load the ARM simulator, it will open in the same state you closed it (i.e., the project and source file loaded). If the project is not open, select the Project tag, and then select **Open Project.** from the pull down window. If you are not in the correct directory, select the appropriate directory in the normal way. Then click on **MyFirstExample.uvproj** which is the name of the project we set up, and then click on the **Open** tab. This loads your project and you are ready to go.

To run the code select **Debug** from the top menu. From the pull down menu select **Start/Stop Debug Session**. This brings up a message telling you that you are in the **EVALUATION MODE** and you click **OK** to dismiss it. You should see a screen like the one below. You can operate on it exactly like any other Windows application and use the **View** tab to open other Windows (such as display memory).



Now you can execute code. We are interested in the instruction-by-instruction mode that lets you execute an instruction at a time. If you click on the step-in button you will execute a single instruction.

This is the step-in button.

You will be able to see any changes in the registers on the left. You will also be able to see the value of the program counter, PC, and the status register, CPSR. Note that function key F11 performs the same operation.

When you've finished, you must click on the **Start/Stop Debug Session** item under the **Project** menu. This returns you to the source code which you can change as necessary. After you'd changed it you must use the **Build target** command again to perform a re-assembly.

Example 2 ADDITION

This problem is the same as Example 1. P = Q + R + SOnce again, let Q = 2, R = 4, S = 5 and assume r1 = Q, r2 = R, r3 = S. In this case, we will put the data in memory in the form of constants before the program runs.

The Code

MOV r1,#Q ;load Q into r1
MOV r2,#R ;load R into r2
MOV r3,#S ;load S into r3
ADD r0,r1,r2 ;Add Q to R
ADD r0,r0,r3 ;Add S to (Q + R)

Here we use the instruction MOV that copies a value into a register. The value may be the contents of another register or a literal. The literal is denoted by the # symbol. We can write, for example, MOV **r7**, r0, MOV **r1**, #25 or MOV **r5**, #Time

We have used symbolic names Q, R and S. We have to relate these names to actual values. We do this with the EQU (equate) assembler directive; for example,

Q EQU 2

Relates the name Q to the value 5. If the programmer uses Q in an expression, it is exactly the same as writing 2. The purpose of using Q rather than 2 is to make the program more readable.

The program

AREA Example2, CODE, READONLY ;load r1 with the constant Q MOV r1,#Q MOV r2,#R MOV r3,#S ADD r0,r1,r2 ADD r0,r0,r3 Stop В Stop Q EQU ;Equate the symbolic name Q to the value 2 2 EQU R 4 ; 5 S EQU ; END

Figure Example 2.1 shows the state of the system after the code has been loaded. If you look at the disassembly window, you will see that the constants have been replaced by their actual values.

Figure Example 2.2 shows the situation after the code has been executed.

Figure Example 2.1 The state of the system after loading the code.

E:\0	CengageEditi	ion_2\Structur	edAR	Mexam	ples\Exa	nple2_/	ADDITIO	Na.uvp	oroj - µV	ision4			- D ×
<u>F</u> ile	<u>E</u> dit <u>V</u> iew	<u>P</u> roject Fl <u>a</u> sh	ח <u>D</u>	ebug f	Pe <u>r</u> ipheral	i <u>T</u> ool	s <u>S</u> VCS	<u>W</u> in	dow <u>H</u>	elp			
1 🗋 🛛	i 🗐 🛃	X 电图 4			> 1ª 1	19 B	微 律	軍//	1 // ž 1	<u>Þ</u>	• 🔊 🖑	۹ 🌢	ି 🔗 💰
RST	El 🛞 🤻	} {} {} {} {} {} {} {} {} {} {} {} {} {}	\$	> 🔍	I I	à 👼	- 📰 -	-	🔜 👻 🗄	🗉 + 🚾 + 🎌 + 🗏			
Registe	ers	ť	ιx	Disasse	mbly								ĻΧ
Regist	ter	Value			2:		MOV :	r1,#Q	; ;1	load r1 with the	e constant Q		
	urrent			<mark><></mark> 0x0	0000000) E37	101002	MOV	r	R1,#0x0000002	2		
	R0	0x00000000			3:		MOV	r2,#R					
	R1	0x00000000		0x0	0000004	E3A	102004	MOV		R2,#0x0000000	1		
	····· R2	0x00000000		0.00	4: 0000000		MOV :	r3,#3 MOU		D2 #0-0000000			
	····· R3	0x00000000			5.	D ESF	103003	r0 r1	r 2	K3,#0X0000000	,		
	····· R4	0x0000000		0x0	0000000	E E DE	310002	ADD)	R0.R1.R2			
	R5	0x0000000			6:		ADD :	r0,r0	,r3	,			
	R6 D7	0x00000000		0x0	0000010) E08	800003	ADD	,	R0,R0,R3			
	Π/ 	0x00000000			7: St	op	в	Stop					
	R9	0x00000000		0x0	0000014	EAF	FFFFE	в		0x0000014			
	R10	0x00000000		0x0	0000018	3 000	00000	AND	EQ	R0,R0,R0			_
		0x00000000											•
	R12	0x00000000			Example	2.5							▼ X
	····· R13 (SP)	0x00000000					-	1.0	0000	22220111			
	R14 (LR)	0x0000000			1	AREA	Examp	piez,	CODE,	READONLY			
	····· R15 (PC)	0x00000000			1	MOV	r2 #1	2	;1040	ri with the con	istant Q		
	CPSR	0x00000003		0.0		MOV	r3.#	5					
	ar/Svetam	0x0000000		05		ADD	r0, r1	1, r2					
	ast Internut			06		ADD	r0, r(), r3					
	temupt			07	Stop	в	Stop						
	upervisor			08									
	bort			09	Q	EQU	2		;Equat	te the symbolic	name Q to th	he valu	e 2
U 🕂 🕂	ndefined			10	R	EQU	4		2 - C				
🛛 🗆 🗝 In	temal			11	S	EQU	5		£				
	PC \$	0×00000000	•	12		END							
🖭 Pro	oject 🛛 🗮 Reg	jisters											▶
													Simulat //

Figure Example 2.2 The state of the system after running the code.

🔣 E:\Cen	gageEditi	on_2\Stri	ucturedA	RMexam	ples\Exam	ple2_/	DDITIONa	.uvproj -	µVision	14			
<u>F</u> ile <u>E</u> d	it <u>V</u> iew	<u>P</u> roject	Fl <u>a</u> sh	<u>D</u> ebug	Pe <u>r</u> ipherals	Tools	<u>s</u> vcs	<u>W</u> indow	<u>H</u> elp				
I 🗋 💕		8 B C	3 9	e (~)		1.121		≓ //≞ //⊼	2		- 🗟 🕫	ه اي	000
RST 🗉	(1) 🛞 🕽	• 🔂 🚯	*{}	5		5 👼 ·	- 📰 - 📴	2 - 👿 -		👿 • 🔆 • 📃			
Registers			д×	Disasse	embly								џх
Register		Value		1	2:		MOV r1	,#Q	;load	d r1 with the	constant	Q	
	ent	10.00		0x0	0000000	E3A	01002	MOV	R1	1,#0x00000002			
	RO	0x0000	000B		3:		MOV r2	,#R					
F	R1	0x0000	0002	0x0	0000004	E3A	02004	MOV	R2	2,#0x00000004			
F F	32	0x0000	0004		4:		MOV r3	,#S					
F	3	0x0000	0005	UX0	000008	E 3A	103005	MOV	R	3,#0 x 000000005			
F	R4	0x0000	0000		000000	FOR	ADD 10 10002	, 11, 12 ADD	R	1 81 82			
F	R5	0x0000	0000		6:	200	ADD r0	.r0.r3		5,11,12			
	R6	0x0000	0000	0x0	0000010	E08	00003	ADD	R	0,R0,R3			
	1/	0x0000	0000		7: Sto	qq	B St	op					
	10	0x0000	0000	<mark> =</mark> 0x0	0000014	EAF	FFFFE	В	02	x0000014			
	R10	0x0000	0000	0x0	0000018	000	00000	ANDEQ	R	D,RO,RO			-
	R11	0x0000	0000										►
F	R12	0x0000	0000		Evample	•							• ×
F	R13 (SP)	0x0000	0000										
F	R14 (LR)	0x0000	0000	01		AREA	Exampl	e2, COI	DE, RE	CADONLY			
F	R15 (PC)	0x0000	0014	02		MOV	r1,#Q	;108	ad rl	with the cons	stant Q		
	PSR	0x0000	00D3	0.3		MOV	$r_{2,\#R}$						
	PSR /Curters	0x0000	0000	04		ADD	r0.r1.	r2					
User/	oystem			06		ADD	r0,r0,	r3					
+ Inter	unt			→ 07	Stop	в	Stop						
E Supe	ervisor			80									
				09	Q	EQU	2	;Equ	iate t	the symbolic r	name Q to	the valu	1e 2 🔜
🗄 🗄 \cdots Unde	fined			10	R	EQU	4	2					
Inter	nal		-	11	S	EQU	5	7					
				12	1	END							
E Projec	кед	isters			J								
													Simulat

Alan Clements ARM simulator notes

Example 3 ADDITION

The problem once again is P = Q + R + S. As before, Q = 2, R = 4, S = 5 and we assume that r1 = Q, r2 = R, r3 = S.

In this case, we will put the data in memory as constants before the program runs. We first use the load register, $LDR \ r1, Q$ instruction to load register r1 with the contents of memory location Q. This instruction *does not exist* and is not part of the ARM's instruction set. However, the ARM assembler automatically changes it into an actual instruction. We call LDR r1, Q a *pseudoinstruction* because it behaves like a real instruction. It is indented to make the life of a programmer happier by providing a shortcut.

The Code

```
LDR
      r1,Q
                ; load r1 with Q
      r2,R
LDR
               ;load r2 with R
               ;load r3 with S
LDR
      r3,S
ADD
      r0,r1,r2 ;add Q to R
ADD
      r0,r0,r3 ;add in S
STR
      r0,Q
               ;store result in Q
```

The program

	AREA	Example3,	CODE, READWRITE
	LDR	r1,Q	;load r1 with Q
	LDR	r2 , R	;load r2 with R
	LDR	r3,S	;load r3 with S
	ADD	r0,r1,r2	;add Q to R
	ADD	r0,r3	;add in S
	STR	r0 , Q	;store result in Q
Stop	В	Stop	
	AREA	Example3,	CODE, READWRITE
P	SPACE	4	;save one word of storage
Q	DCD	2	;create variable Q with initial value 2
R	DCD	4	;create variable R with initial value 4
S	DCD	5	;create variable S with initial value 5
	END		

Note how we have to create a data area at the end of the program. We have reserved spaces for P, Q, R, and S. We use the SPACE directive for S to reserve 4 bytes of memory space for the variable S. After that we reserve space for Q, R, and S. In each case we use a DCD assembler directive to reserve a word location (4 bytes) and to initialize it. For example,

Q DCD 2 ;create variable Q with initial value 2

means 'call the current line Q and store the word 0x00000002 at that location.

Figure Example 3.1 shows the state of the program after it has been loaded. In this case we've used the view memory command to show the memory space. We have highlighted the three constants that have been pre-loaded into memory.

Take a look at the disassembled code. The pseudoinstruction LDR r1, Q was actually translated into the real ARM instruction LDR r1, [PC, #0x0018]. This is still a load instruction but the addressing mode is register indirect. In this case, the address is the contents of the program counter, PC, plus the hexadecimal offset 0x18. Note also that the program counter is always 8 bytes beyond the address of the current instruction. This is a feature of the ARM's pipeline.

Consequently, the address of the operand is [PC] + 0x18 + 8 = 0 + 18 + 8 = 0x20.

If you look at the memory display area you will find that the contents of 0x20 are indeed 0x00000002.

Figure Example 3.1 The state of the system after loading the program

🖫 E:\CengageEdit	ion_2\Structured	IARMexamples\Example3_ADDITIONb.uvproj - µVision4	×
<u>F</u> ile <u>E</u> dit <u>V</u> iew	<u>P</u> roject Fl <u>a</u> sh	<u>D</u> ebug Pe <u>r</u> ipherals <u>T</u> ools <u>S</u> VCS <u>W</u> indow <u>H</u> elp	
🗋 🚄 🛃 🥔	るも色 ク	▶ ◇ ◇ 教務務 律準准版 🙆 💿 🕄 🔍 ● ○ 🖒	2
👫 🗐 🐼 🤻	} (<u>)</u> +() +() +() +()	>	
Registers	τ×	Disassembly P	×
Register	Value	2: LDR r1,Q ;load r1 with Q	<u> </u>
		C)0x00000000 E59F1018 LDR R1, [PC, #0x0018]	
RO	0x00000000	0x00000004 E59F2018 LDR R2, [PC, #0x0018]	
R1	0x00000000	4: LDR r3,S ;load r3 with S	
	0x00000000	0x00000008 E59F3018 LDR R3, [PC, #0x0018]	
R4	0x0000000	5: ADD r0,r1,r2 ; add Q to R	
R5	0x0000000	6: ADD r0,r3 ;add in S	
R6	0x00000000	0x00000010 E0800003 ADD R0, R0, R3 The code generate	ed by
	0x00000000	7: STR r0,Q ;store result in Q the pseudoinstruc	tion
R9	0x0000000	0x00000014 E58F0004 STR R0, [PC, #0x0004] LDR r1, Q.	
R10	0x00000000	0x00000018 EAFFFFE B 0x00000018	
R12	0x0000000	0x0000001C 00000000 ANDEQ R0,R0,R0	-
R13 (SP)	0x00000000		
····· R14 (LR)	0x0000000	Example3.s	×
R15 (PC)	0x00000000	01 AREA Example3, CODE, READWRITE	F
E SPSR	0x00000003	D2 LDR r1,Q ;load r1 with Q	-
⊞ User/System	0.0000000	03 LDR r2, R ;load r2 with R	
+ Fast Interrupt		04 LDR r3,S ;load r3 with S	
Interrupt		05 ADD $r0, r1, r2$; add Q to R	
		07 STB r0.0 store result in 0	
		08 Stop B Stop	
E Internal		09	
PC \$	0x00000000	10 AREA Example3, CODE, READWRITE	
Mode	Supervisor	11 P SPACE 4 ;save one word of storage	
States	0	13 R DCD 4 :create variable R with initial value 4	
Sec	0.0000000	14 S DCD 5 ;create variable S with initial value 5	
		15 END	-
🖭 Project 🧮 Reg	jisters		
Memory 1		4	×
Address: 0		· · · · · · · · · · · · · · · · ·	-
0x00000000: E	5 9F 10 18 E	25 9F 20 18 E5 9F 30 18 E0 81 00 02 E0 80 00 03 E5 8F 00 04 EA FF FF FE	
0x0000001C: 0	0 00 00 00 0	00 00 00 02 00 00 04 00 00 05 00 00 00 00 00 00 00 00 00 00	-1
0x0000038: 0			<u> </u>
Call Stack + Loca	als Menfory 1		
		Simu	14
The address of	of the first data	a element on These are the three data values we've	
this line is 0x	0000001C. Th	he first stored in memory at locations	
element of the	e next word (i.	.e., the fifth	
byte in the see	quence) is at a	address 0x0000020	
0x0000001C	+4 = 0x00000	0020. 0x00000024	
		0x0000028	
		These locations are chosen by the	
		assembler automatically.	

R:	:\CengageEdit	ion_2\StructuredARI	RMexamples\Example3_ADDITIONb.uvproj - µVision4	<u> </u>
<u>F</u> ile	<u>E</u> dit <u>V</u> iew	<u>P</u> roject Fl <u>a</u> sh <u>D</u> e	ebug Pe <u>r</u> ipherals <u>T</u> ools <u>S</u> VCS <u>W</u> indow <u>H</u> elp	
	💕 🛃 🦪	多時間 タウ	- 🖉 👳 🦉 🍇 🊎 筭 /// [6] 🤷 💽 💽 🍕 👰 🔶 🤇	ି 🔗 🖪
RST	F 🗉 🛞 🛛 🖥	ት በት ⊕ +() ቀበ የ	N 🔯 📑 🖓 - 🔜 - 🔜 - 🔜 - 🕺 - 🔄	
Regi	sters	д×	X Disassembly	μ Χ
Reg	gister	Value	2: LDR r1,Q ;load r1 with Q	_
	Current		- 0x00000000 E59F1018 LDR R1, [PC, #0x0018] 3: LDR r2 R :load r2 with R	
	RO	0x0000000	0x00000004 E59F2018 LDR R2, [PC, #0x0018]	
	RI D2	0x0000002	4: LDR r3,S ;load r3 with S	
	R2	0x0000004	x00000008 E59F3018 LDR R3,[PC,#0x0018]	
	R4	0x00000000	5: ADD r0,r1,r2 ;add Q to R	
		0x00000000	➡Ox0000000C E0810002 ADD R0,R1,R2	
	R6	0x00000000	6: ADD r0,r0,r3 ;add in S	
	R7	0x00000000	0x00000010 E0800003 ADD R0,R0,R3	
	R8	0x0000000	7: SIR FU,Q ;Store result in Q	
	R9	0x00000000	8: Stop B Stop	
	R10	0x00000000	0x00000018 EAFFFFE B 0x00000018	
	R11	0x0000000	0x0000001C 00000000 ANDEQ R0,R0,R0	-
	R12	0x0000000		•
	B14 (LR)	0x00000000	T Example2 c	- ×
	R15 (PC)	0x0000000C		<u> </u>
	E CPSR	0x00000D3	01 AREA Example3, CODE, READWRITE	
		0x00000000	U2 LDR r1, g ; load r1 with g	
÷	User/System		DR r_2 , r_3 r_2 $vith R$	
+	Fast Interrupt		r_{0} ADD r_{0} , r_{1} , r_{2} and r_{0} to R	
1 ± ·····	Interrupt		06 ADD r0,r0,r3 ;add in S	
	Supervisor		07 STR r0,Q ;store result in Q	
	Undefined		08 Stop B Stop	
	Internal		09	
	PC \$	0x000000C	10 AREA Example3, CODE, READWRITE	
	····· Mode	Supervisor	11 P SPACE 4 ;save one word of storage	
	····· States	9	12 P DCD 2 ;create variable g vith initial value	2
	Sec	0.00000000	14 S DCD 5 create variable S with initial value	4
			15 END	Ŭ 🔳
E P	Project 🚟 Reg	gisters		
Mem	iory 1			 μ×
Add	dress: 0			
0x0	0000000: E	5 9F 10 18 E5 9	9F 20 18 E5 9F 30 18 E0 81 00 02 E0 80 00 03 E5 8F 00 04 EA FF FF FE	00
0x0	000001D: 0	0 00 00 00 00 00	00 02 00 00 04 00 00 05 00 00 00 00 00 00 00 00 00 00	00
	0000037 • 0			
¢0 (Lall Stack + Loc	als Memory 1		
			S	imulat 🏼 🎢

Figure Example 3.2 The state of the system after loading the program

Figure Example 3.3 The state of the system after loading the program

R:	\CengageEditi	ion_2\Structured#	dARMexamples\Example3_ADDITIONb.uvproj - μVision4	
<u>F</u> ile	<u>E</u> dit <u>V</u> iew	<u>P</u> roject Fl <u>a</u> sh	<u>D</u> ebug Pe <u>r</u> ipherals <u>T</u> ools <u>S</u> VCS <u>W</u> indow <u>H</u> elp	
	💕 🛃 🗿	る 単語 ター	◇ ◇ ◎ 懇 為 我 準 準 /// 版 ❷ ■ ■ & Ø ●	୦ 🔗 🛃
RST	1 🛛 🕹	} () +() +() +() +()	> 🖸 🔯 🖬 🖓 - 🔲 - 😨 - 📾 - 💷 - 💥 - 🔍	
Regis	ters	ф	a X Disassembly	ŢХ
Regi	ister	Value	2: LDR r1,Q ;load r1 with Q	
	Current		Ox0000000 E59F1018 LDR R1, [PC, #0x0018]	
	R0	0×000000B	3: LDR r2,R ;load r2 with R	
	R1	0x0000002	4. LDR r3 S :load r3 with S	
	R2	0x0000004	0x00000008 E59F3018 LDR R3. [PC.#0x0018]	
	R3	0x00000005	5: ADD r0,r1,r2 ;add 0 to R	
	R4	0x0000000	0x0000000C E0810002 ADD R0,R1,R2	
	R9 DC	0x00000000	6: ADD r0,r0,r3 ;add in S	
	R7	0x00000000	0x00000010 E0800003 ADD R0,R0,R3	
	R8	0x00000000	7: STR r0,Q ;store result in Q	
	R9	0x00000000	0x00000014 E58F0004 STR R0,[PC,#0x0004]	
	R10	0x00000000	8: Stop B Stop	
	R11	0x00000000	CX00000018 EAFFFFE B 0X00000018	
	R12	0x00000000	VXUUUUUUU ANDEQ RU, RU, RU	الشريب الم
	R13 (SP)	0x00000000		
	R14 (LR)	0x0000000	Example3.s	▼ ×
	R15 (PC)	0x00000018	01 AREA Example3, CODE, READWRITE	<u> </u>
		0x00000D3	02 LDR r1,Q ;load r1 with Q	-
	⊡‴ SFSR Lleer/Suttem	0x0000000	03 LDR r2,R ;load r2 with R	
÷ + ·····	East Interrupt		04 LDR r3,S ;load r3 with S	
÷ +	Internation		05 ADD r0,r1,r2 ;add Q to R	
	Supervisor		06 ADD r0,r0,r3 ;add in S	
÷	Abort		07 STR r0,Q ;store result in Q	
÷	Undefined		CO8 βtop B Stop	
÷	Internal		10 APEA Example 2 CODE DEADNETTE	
	PC \$	0x0000018	11 P SPACE 4 realized one word of storage	
	Mode	Supervisor	12 0 DCD 2 create variable 0 with initial value	. 2
	States	16	13 R DCD 4 ;create variable R with initial value	e 4
	Sec	0.00000000	14 S DCD 5 ;create variable S with initial value	e 5
<u> </u>			15 END	
PI	roject 🚟 Reg	isters		
Memo	ory 1			д х
Add	ress: 0			
0x00	0000000: E	5 9F 10 18 E5	25 9F 20 18 E5 9F 30 18 E0 81 00 02 E0 80 00 03 E5 8F 00 04 EA FF FF FI	E 00
0x00	000001D: 0	0 00 00 00 00	<mark>ю оо ов</mark> оо оо оо о4 оо оо оо о5 оо оо оо оо оо оо оо оо оо	0 00 🗐
10v00	000037 0			
Q C	all Stack + LOCA	ins internory 1		
				Simulat //

After executing the program the sum of Q, R, and S has been stored in location P in memory.

Example 4 ADDITION

The problem

P = Q + R + S where Q = 2, R = 4, S = 5. In this case we are going to use register indirect addressing to access the variables. That is, we have to set up a pointer to the variables and access them via this pointer.

The Code

ADR	r4, TheData	;r4 points to the data area
LDR	r1, [r4,#Q]	;load Q into r1
LDR	r2, [r4,#R]	;load R into r2
LDR	r3, [r4,#S]	;load S into r3
ADD	r0 , r1, r2	;add Q and R
ADD	r0, r0 , r3	;add S to the total
STR	r0,[r4,#P]	;save the result in memory

The program

	AREA ENTRY	Example4, COD	DE, READWRITE
	ADR	r4, TheData	;r4 points to the data area
	LDR	r1 , [r4, #0]	;load O into r1
	LDR	r2 , [r4, #R]	;load R into r2
	LDR	r3 , [r4, #S]	;load S into r3
	ADD	r0 , r1, r2	; add O and R
	ADD	r0 , r0, r3	;add S to the total
	STR	r0 , [r4, #P]	; save the result in memory
Stop	В	Stop	,
P	EQU	0	;offset for P
Q	EQU	4	;offset for Q
R	EQU	8	;offset for R
S	EQU	12	;offset for S
	AREA I	Example4, CODE	, READWRITE
TheData	SPACE	4	; save one word of storage for P
	DCD	2	;create variable Q with initial value
	DCD	4	;create variable R with initial value
	DCD	5	;create variable S with initial value
	END		

Figure Example 4.1 shows the state of the system after the program has been loaded.

I have to admit, that I would not write this code as it is presented. It is far too verbose. However, it does illustrate several concepts.

First, the instruction ADR r4, TheData loads the address of the data region (that we have labelled TheData into register r4. That is, r4 is pointing at the data area. If you look at the code, we have reserved four bytes for P and then have loaded the values for Q, R and S into consecutive word location. Note that we have not labelled any of these locations.

The instruction ADR (load an address into a register) is a pseudoinstruction. If you look at the actual disassembled code in Figure Example 4.1 you will see that this instruction is translated into ADD r4, pc, #0x18. Instead of loading the actual address of TheData into r4 it is loading the PC plus an offset that will give the appropriate value. Fortunately, programmers can sleep soundly without worrying about how the ARM is going to translate an ADR into actual code – that's the beauty of pseudoinstructions.

When we load Q into r1 we use LDR r1, [r4, #Q]. This is an ARM load register instruction with a literal offset; that is, Q. If you look at the EQU region, Q is equated to 4 and therefore register r1 is loaded with the data value that is 4 bytes on from where r4 is pointing. This location is, of course, where the data corresponding to Q has been stored.

2 4 5 Figure Example 4.1 The state of the system after loading the program

🔣 E:\CengageEditi	on_2\Structured/	Mexamples\Example4_ADDITIONc.uvproj - µVision4	
<u>F</u> ile <u>E</u> dit <u>V</u> iew	<u>P</u> roject Fl <u>a</u> sh	ebug Pe <u>r</u> ipherals <u>T</u> ools <u>S</u> VCS <u>W</u> indow <u>H</u> elp	
I 🗋 💕 🛃 🗿	み 山 臨 🥱	◇ - ◇ 巻 祭 祭 御 華 旭 版 🙆 🔹 🔹 🕺 🔍 🌢 ○ ⊘ 🎎	- 1
	- 0° -0 -0 -		
Registers	<u>.</u>	Disassembly	ąх
Begister	Value	3: ADR r4,TheData ;r4 points to the data area	
	10.00	<pre>c>0x00000000 E28F4018 ADD R4, PC, #0x00000018</pre>	
R0	0x0000000	4: LDK r1,[r4,#Q] ;load Q into r1 0x00000004 E5941004 LDR R1.[R4,#0x0004]	
R1	0x00000000	5: LDR r2,[r4,#R] ;load R into r2	
R3	0x00000000	0x00000008 E5942008 LDR R2, [R4, #0x0008]	
R4	0x00000000	6: LDR r3, [r4, #S] ; load S into r3	
R5	0×0000000	7: ADD r0.r1.r2 :add 0 and R	
R6	0x00000000	0x00000010 E0810002 ADD R0,R1,R2	
R8	0x00000000	8: ADD r0,r0,r3 ;add S to the total	
R9	0x00000000	0x00000014 E0800003 ADD R0,R0,R3	
R10	0x00000000	0x00000018 E5840000 STR R0, [R4]	
B12	0x00000000	10: Stop B Stop	
R13 (SP)	0x00000000	0x000001C EAFFFFE B 0x000001C	
R14 (LR)	0x0000000	0x00000020 00000000 ANDEQ R0,R0,R0	
R15 (PC)	0x00000000	0x00000028 0000004 ANDEQ R0,R0,R4	
E SPSR	0x00000003	0x0000002C 00000005 ANDEQ R0,R0,R5	-
User/System			
Fast Interrupt		Example4.s	• ×
		01 AREA Example4, CODE, READWRITE	=
		02 ENTRY	
E Undefined		C)03 ADR r4, TheData ;r4 points to the data area	
	0-0000000	104 LDR r1, [r4, $\frac{1}{7}$]; load Q into r1 105 LDR r2, [r4, $\frac{1}{7}$ R]; load R into r2	
Mode	Supervisor	06 LDR r3, [r4, #S] ; load S into r3	
States	0	07 ADD r0,r1,r2 ;add Q and R	
Sec	0.0000000	08 ADD r0,r0,r3 ;add S to the total	
		10 Stop B Stop	
		11	
		12 P EQU 0 ;offset for P	
		13 Q EQU 4 ;offset for Q	
		15 S EOU 12 coffset for S	
		16	
		17 AREA Example4, CODE, READWRITE	
		18 IneData SPACE 4 ;save one word of storage for P 19 DCD 2 :create wariable 0 with initial value 2	
		20 DCD 4 ;create variable R with initial value 4	
		21 DCD 5 ;create variable S with initial value 5	
		22 END	
🖭 Project 🧮 Regi	isters		
Memory 1			ąχ
Address: 0			
0x00000000: E2	2 8F 40 18 E	94 10 04 E5 94 20 08 E5 94 30 0C E0 81 00 02 E0 80 00 03 E5 84 00 00 EA FF F	FF
0x0000001F: FI	E 00 00 00 00	00 00 00 02 00 00 00 04 00 00 00 05 00 00 00 00 00 00 00 00 00	00
0x0000003E: 00			
Call Stack + Loca	ls Memory 1		
		Simulation	

Figure Example 4.2 The state of the system after executing the program

E:\CengageEdit	tion_2\Structured/	Mexamples\Example4_ADDITIONc.uvproj - µVision4	- 🗆 🗵
<u>F</u> ile <u>E</u> dit <u>V</u> iew	<u>P</u> roject Fl <u>a</u> sh	bug Pe <u>r</u> ipherals <u>T</u> ools <u>S</u> VCS <u>W</u> indow <u>H</u> elp	
🗋 💕 🖌 🥔	る時間の	◇ ◇ 巻 巻 巻 集 拝 振 💩 ▼ ■ ▲ グ 🔍 ● ○ 🔗	👧 🖃 🗖
	<u>, , , , , , , , , , , , , , , , , , , </u>		
Registers	л 1000 г		лх
Registers	Value	3: ADR r4.TheData :r4 points to the data area	* ^
Register	Value	0x00000000 E28F4018 ADD R4, PC, #0x00000018	
R0	0x000000B	4: LDR r1, [r4, #Q] ;load Q into r1	
R1	0x0000002	0x00000004 E5941004 LDR R1, [R4, #0x0004]	
R2	0x0000004	0x00000008 E5942008 LDR R2, [R4, #0x0008]	
R3	0x0000005	6: LDR r3, [r4, #S] ;load S into r3	
R5	0x00000000	0x000000C E594300C LDR R3, [R4, #0x000C]	
R6	0x00000000	7: ADD r0,r1,r2 ;add Q and R	
R7	0x0000000	8: ADD r0.r0.r3 :add S to the total	
R8	0x00000000	0x00000014 E0800003 ADD R0,R0,R3	
R10	0x00000000	9: STR r0,[r4,#P] ;save the result in memory	
R11	0x00000000	0x00000018 E5840000 STR R0, [R4]	
R12	0x00000000	10: Stop B Stop	
R13 (SP)	0x00000000	0x00000020 0000000B ANDEQ R0,R0,R11	
R14 (LR)	0x00000000	0x00000024 00000002 ANDEQ R0,R0,R2	
	0x00000D3	0x00000028 00000004 ANDEQ R0,R0,R4	
	0x00000000	0x0000002C 00000005 ANDEQ R0,R0,R5	
± User/System			
them.pt		Example4.s	▼ ×
E Supervisor		01 AREA Example4, CODE, READWRITE	
Abort		02 ENTRY	
		ADR r4, IneData ;r4 points to the data area	
PC \$	0x0000001C	05 LDR r2, [r4, #R] ; load R into r2	
Mode	Supervisor	06 LDR r3,[r4,#S] ;load S into r3	
States	14	07 ADD r0,r1,r2 ;add Q and R	
Sec .	0.00000000	09 STR r0.[r4.#P1 :save the result in memory	
		10 Stop B Stop	
		11	
		12 P EQU 0 ;offset for P	
		13 Q EQU 4 ;offset for Q	
		15 S EQU 12 ;offset for S	
		16	
		17 AREA Example4, CODE, READWRITE	
		18 IneData SPACE 4 ;save one word of storage for P 19 DCD 2 :create variable 0 with initial value	2
		20 DCD 4 ;create variable R with initial value	4
		21 DCD 5 ;create variable S with initial value	5 —
L		22 END	_
🖻 Project 🗮 Reg	gisters		
Memory 1			 μ×
Address: 0			
0x0000000: H	2 8F 40 18 E	94 10 04 E5 94 20 08 E5 94 30 0C E0 81 00 02 E0 80 00 03 E5 84 00 00 EA	FF FF
0x0000001F: H	TE 00 00 00 01	00 00 00 02 00 00 04 00 00 05 00 00 00 00 00 00 00 00 00 00	00 00
0x0000003E: 0			
Call Stack + Loc	als Memory 1		
		Simulat	ion //

Example 5 ADDITION

We're going to repeat the same example once again. This time we will write the program in a more compact fashion, still using the ADR (load register with address instruction).

To simplify the code, we've used simple numeric offsets (because there is relatively little data and the user comments tell us what's happening. Note that we have used labels Q, R, and S for the data. These labels are redundant and are not needed since they are not referred to anywhere else in the program. There's nothing wrong with this. These labels just serve as a reminder to the programmer.

```
Example5, CODE, READWRITE
        AREA
        ENTRY
        ADR
              r0, P
                            ;r4 points to the data area
        LDR
              r1, [r0, #4]
                            ;load Q into r1
        LDR
              r2, [r0, #8]
                            ;load R into r2
        ADD
              r2, r1, r2
                            ;add Q and R
        LDR
              r1, [r0, #12] ; load S into r3
        ADD
              r2, r2, r1
                            ;add S to the total
        STR
              r1, [r2]
                            ; save the result in memory
Stop
        В
              Stop
        AREA Example5, CODE, READWRITE
Ρ
        SPACE 4
                            ;save one word of storage for P
0
        DCD
              2
                            ;create variable Q with initial value 2
R
        DCD
              4
                            ;create variable R with initial value 4
S
        DCD
              5
                            ;create variable S with initial value 5
        END
```

Note also that we have reused registers to avoid taking up so many. This example uses only r0, r1, and r2. Once a register has been used (and its value plays no further part in a program, it can be reused. However, this can make debugging harder. In this example at one point r1 contains Q and at another point it contains S. Finally, it contains the result S.

Figure Example 5.1 gives a snapshot of the system after the program has been loaded, and Figure Example 5.2 shows the state after the program has been executed.

🔣 E:	\Cengag	eEditi	on_2\St	ructure	edAF	RMexa	mples\Exa	mple5_	ADDI	TIONd.	uvproj	- μV	ision4											×
<u>F</u> ile	<u>E</u> dit	<u>V</u> iew	<u>P</u> roject	Flash	D D	ebug	Pe <u>r</u> iphera	ls <u>T</u> oo	ls <u>S</u>	vcs <u>v</u>	<u>V</u> indow	/ <u>H</u>	elp											
	🞽 🛃	9	a Pal				- ba	厚悶	12		/= /i	億日	2				- 5	l Car	0	•	0	ଚ 💰] -
RST	E. Ø) [}	• 🔂 🕀	*{}	♦	>	à 🖪 🚍	s 🖉	-	• 🛃	- 🔜	- 🗄	💷 👻 🧱	-	🏷 - 🔨	1								
Regi	sters			д 2	×	Disasse	mbly																ą	чx
Reg	jister		Value		7		3:		ADR	r0	, P		;r	4 po	ints	to t	he da	ita a	rea					-
-	Current		_			\$0x0(0000000 4۰	E281	T.DR	AD:	D [20]	#41	R0, PC,	,#0x	00000	018 0 r1								-
	R0		0x000	00000		0x0	0000004	E590	1004	LD	R 1107	π + J	R1, [R	0,#0	x0004]								
			0x000	000000			5:		LDR	r2	,[r0,	#8]	;10	ad	R int	o r2								
	R3		0x000	00000		0x0	6.000008	E590	2008 מתג	1 LD	R 71 7	2	R2,[R0	0,#0	x0008]								
	R4		0x000	00000		0x0	0000000	E081	2002	AD	D	2	R2,R1,	, R2	ana									
	R6		0x000	000000			7:		LDR	r1	,[r0,	#12] ;10	ad	S int	o r3								
	R7		0x000	00000		0x0	0000010	E590	1000	: LDI	R 	1	R1, [R0	D,#0	x000C]	ot o 1							
			0x000	00000		0x0	0000014	E082	2001	. AD	, <u>r</u> 2, r D	1	R2,R2	, R1	10 1	ne u	OUAL							
	R10		0x000	00000			9:		STR	r1	,[r0]		; 38	ave	the r	esul	t in	memo	ry					
	R11		0x000	00000		0x0	0000018	E580	1000) STI	R		R1,[R0	[[
	R12		0x000	00000		0x0	10: Sto 000001C	p EAFF	B	B B	op		0x000	0001	с									
	R13	(SP) (LR)	0x000	00000		0x0	0000020	0000	00000) AN	DEQ		RO,RO	, R0										
	R15	(PC)	0x000	000000		0x0	0000024	0000	00002	AN	DEQ		RO,RO	, R2										
	E CPSF	R	0x000	000D3		0x0	0000028 000002C	0000	00004	AN AN	DEQ DEO		RO,RO, RO,RO,	, R4 . R5										
 	±···· SPSF User/Svet	tem	0x000	00000		• <u>()</u>																		- -
	Fast Inten	rupt								<u>ה</u>														_
	Interrupt				H		Examples		ION.s															×
+	Supervi: Abort	SOL				01		AREA	A Ex	ample	≥5, C	ODE	, REAI	DWRI	TE									
	Undefined	Ь				<202		ADR	r0	,P		1	r4 po:	ints	to t	he d	lata a	rea						
<u> </u>	Internal					04		LDR	r1	,[r0	,#4]	1	load (2 in	to r1									
	PC \$		0x000 Super	00000 Visor		05		LDR	r2	,[r0,	,#8] r2	1	load H	R in	to r2									
	State	s	0	1301		07		LDR	r1	,[r0	,#12]	4	load S	s in	to r3									
	Sec		0.000	00000		08		ADD	r 2	, r2,	r1	1	add S	to	the t	otal								
						09	Stop	STR	r1	,[r0]]	1	save t	the	resul	t in	memo	ry						
						11	JUUD	5	50	.qu														
						12		AREA	Exa	mple	5, CO	DE,	READ	RIT	E									
						13	P	SPAC DCD	E 4			1	save o	one	word	of s	torag	re fo	r P			2		
						15	R	DCD	4			4	create	e va	riabl	e R	with	init	ial	val	ue 4	4		
						16	S	DCD	5			1	create	e va	riabl	e S	with	init	ial	val	ue !	5		
						17		END																┓
E P	Project	Regi	isters			 • [•	
Mem	ory 1																						Ļ	I X
Add	lress: 0																						L]_
0x0	000000	0: E2	2 8F 0	0 18	E5	90 1	0 04 E5	90 2	0 08	E0 8	1 20	02	E5 90	10	OC EC	82	20 01	1 E5	80	10	00 E	A FF	FF	
0x0	000001	r: FI E: 00	E 00 0 0 00 0	00 00	00	00 0	0 00 02	00 0	0 00	04 0	0 0 0	00	05 00	00	00 00	00 00	00 00	00 0	00	00	0000 000	U 00 0 00	00	
and a	all Stack	+ Loca		1emon/	1	00 0	0000	00 0	5 50	00 0	5 50	00	50 00	00	55 00		55 00		00	55		000	00	
		2000			_																Simu	Ilation		

Figure Example 5.1 The state of the system before executing the program

Figure Example 5.2 The state of the system after executing the program

E:\CengageEdition_2\Structured	ARMexamples\Example5_ADDITIONd.uvproj - µVision4	- I X
<u>File E</u> dit <u>V</u> iew <u>P</u> roject Fl <u>a</u> sh	<u>D</u> ebug Pe <u>r</u> ipherals <u>T</u> ools <u>S</u> VCS <u>W</u> indow <u>H</u> elp	
🗋 😂 🛃 🕼 3 🕒 🖎 9	♀ ◇ ● ❷ 懇 想 事 罪 順 版 ❷ ■ ■ 動 @ ● ● ◇ � !	-
👫 🗉 📀 79 🖓 ()^ *() 🔅	> 2 🔯 🖬 🖓 - 💷 - 😰 - 🗰 - 🔯 - 🔀 - 📃	
Registers 🛛 🗘 🗙	Disassembly	ąΧ
Register Value	3: ADR r0,P ;r4 points to the data area	
	4: LDR r1,[r0,#4] ;load Q into r1	
R0 0x0000020	0x00000004 E5901004 LDR R1,[R0,#0x0004]	
R2 0x000000B	5: LDR r2,[r0,#8] ;load R into r2	
R3 0x0000000	6: ADD r2,r1,r2 ;add Q and R	
R5 0x0000000	0x0000000C E0812002 ADD R2,R1,R2	
R6 0x0000000	7: LDR r1,[r0,#12] ; Ioad S into r3	
R7 0x00000000	8: ADD r2,r2,r1 ;add S to the total	
R9 0x0000000	0x00000014 E0822001 ADD R2,R2,R1	
R10 0x00000000	0x00000018 E5801000 STR R1, [R0]	
R11 0x0000000	10: Stop B Stop	
R13 (SP) 0x00000000	20x0000001C EAFFFFE B 0x0000001C	
R14 (LR) 0x00000000	0x00000024 00000002 ANDEQ R0,R0,R2	
	0x0000028 0000004 ANDEQ R0,R0,R4	
	0x0000002C 00000005 ANDEQ R0,R0,R5	
the set later int		
± Interrupt	Example5_ADDITION.s	▼ ×
	01 AREA Example5, CODE, READWRITE	
	02 ENTRY 03 ADR r0.P :r4 points to the data area	
Internal	04 LDR r1,[r0,#4] ;load Q into r1	
PC \$ 0x0000001C	05 LDR r2, [r0, #8] ; load R into r2	
States 17	07 LDR r1, [r0, #12] ; load S into r3	
Sec 0.0000000	08 ADD r2,r2,r1 ;add S to the total	
	09 STR r1,[r0] ;save the result in memory	
	11	
	12 AREA Example5, CODE, READWRITE	
	13 P SPACE 4 ; save one word of storage for P 14 0 DCD 2 :create variable 0 with initial value 2	
	15 R DCD 4 ;create variable R with initial value 4	
	16 S DCD 5 ;create variable S with initial value 5	
	- 12 END	
E Project Registers		
Memory 1		ф х
Address: 0		
0x00000000: E2 8F 00 18 E	15 90 10 04 E5 90 20 08 E0 81 20 02 E5 90 10 0C E0 82 20 01 E5 80 10 00 EA FF F	(F
0x0000003E: 00 00 00 00 0		
Call Stack + Locals Memory 1		
	Simulation	

Summary – Example 1 to 5

Programs using the Keil ARM IDE begin with a line like AREA nameOfProg, CODE, READONLY and end with END.

- You can store data in memory with the DCD (define constant) before the program runs.
- You can write ADD **r0**, r1, #4 or ADD **r0**, r1, K1. However, if you do use a symbolic name like K1, you have to use an EQU statement to equate it (link it) to its actual value.
- Some instructions are pseudoinstructions. They are not actual ARM instructions but a form of shorthand that is automatically translated into one or more ARM instructions.
- The instruction MOV **r1**, r2 or MOV **r1**, #literal has two operands and moves the contents of a register or a literal into a register.

Example 6 Arithmetic Expressions

The problem

We are going to perform a more complex arithmetic evaluation. Assume that we wish to evaluate (A + 8B + 7C - 27)/4, where A = 25, B = 19, and C = 99.

The Code

We will use literals in this simple example. Note that the ARM has a multiply instruction but no divide instruction.

MOV	r0, #25	;Load register r0 with A which is 25
MOV	r1, #19	;Load register r1 with B which is 19
ADD	r0 , r0, r1, LSL #3	;Add 8 x B to A in r0
MOV	r1, #99	;Load register r1 with C which is 99 (reuse of r1)
MOV	r2, #7	;Load register r2 with 7
MLA	r0 , r1, r2, r0	;Add 7 x C to total in r0
SUB	r0, r0,#27	;Subtract 27 from the total
MOV	r0, r0,ASR #2	;Divide the total by 4

There are two several to note. First, was can multiply or divide by a power-of-2 by shifting left, or right, respectively. Moreover, instructions allow us to perform a shift on the second operand, so that ADD r0, r0, r1, LSL #3 means shift the contents of register r1 left three times (multiply it by 8) and then add it to register r0 and put the total in r0.

Second, we can use the *add and multiply instruction*, MLA, to perform $P = P + Q \cdot R$. In this case we are able to perform the multiplication 7 x C and add that to the running total in r0. Note the format of this instruction.

Finally, we perform the division by 4 moving the result from r0 to r0 while applying two shifts right. Figure Example 6.1 demonstrates the state of the system after the code has been executed.

Figure Example 6.1 The state of the system after executing the program

🔣 E:\Cengag	eEdition_2\Structure	dARMexa	mples\Example6_Ex	cpressions.uvproj - μVisio	on4	_ 🗆 🗙
<u>F</u> ile <u>E</u> dit	<u>V</u> iew <u>P</u> roject Fl <u>a</u> sh	<u>D</u> ebug	Peripherals Tools	<u>S</u> VCS <u>W</u> indow <u>H</u> elp		
🗋 💕 🛃	🍠 🕉 🖻 🛍 💌) 🛯 🤄	⇒ P 22 23 2	🛓 💷 🎼 //編 🕍	- 🗟 🏘 💽 🔶	ି 🔗 🍓 🔳 🔹
🛛 🕅 🕅 🕅	{}* <{} € } {€} {€}	4 🗾 🗹). 🖪 🚍 🦾 🐺 •	🔲 • 🕎 • 🔜 • 💷 •	· 🖬 • 🔆 • 🖳	
Registers		× / [Example6.s			▼ ×
Register R0 R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 R11 R12 R13 R14 (Value 0x00000002 0x0000003 0x0000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000000 0x00000000000000000000000000000000000	▲ 01 02 03 04 05 06 07 08 09 09 10 0 4 11 12 13 14	AREA ENTR MOV MOV ADD MOV MLA SUB MOV S B END	Example6, CODE, Y r0,#25 r1,#19 r0,r0,r1,LSL #3 r1,#99 r2,#7 r0,r1,r2,r0 r0,r0,#27 r0,r0,ASR #2 S	READWRITE ;Load register r0 with A which i ;Load register r1 with B which i ;Add 8 x B to A in r0 ;Load register r1 with C which i ;Load register r2 with 7 ;Add 7 x C to total in r0 ;Subtract 27 from the total ;Divide the total by 4 ;stop here	s 25 s 19 s 99 (reuse of r1)
🖭 Project 🛛	Registers	- Lu				
					S	imulation

Example 7 Logical Operations

Logical operations are virtually the same as arithmetic operations from the programmer's point of view. The principal differences being that logical operations do not create a carry-out (apart from shift operations), and you don't have to worry about negative numbers. Logical operations are called bitwise because they act on individual bits. Basic or fundamental logical operations are:

NOT	Invert bits	$c_i = \overline{a_i}$
AND	Logical and	$c_i = a_i \cdot b_i$
OR	Logical OR	$c_i = a_i + b_i$

Derived logical operations that can be expressed in terms of fundaments operations are (this is not an exhaustive list):

XOR	Exclusive OR	$c_i = \overline{a_i} \cdot b_i + a_i \cdot \overline{b_i}$
NAND	NOT AND	$c_i = \overline{a_i \cdot b_i}$
NOR	NOT OR	$c_i = \overline{a_i + b_i}$

Shift operations are sometimes groups with logical operations and sometime they are not. This is because they are not fundamental Boolean operations but they are operations on bits. A shift operation moves all the bits of a word one or more places left or right. Typical shift operations are:

LSL	Logical shift left	Shift the bits left. A 0 enters at the right hand position and the bit in the left hand position is copied into the carry bit
LSR	Logical shift right	Shift the bits right. A 0 enters at the left hand position and the bit in the right hand position is copied into the carry bit.
ROL	Rotate left	Shift the bits left. The bit shifted out of the left hand position is copied into the right hand position. No bit is lost.
ROR	Rotate right	Shift the bits right. The bit shifted out of the right hand position is copied into the left hand position. No bit is lost.

Some microprocessors include other logical operations. These aren't needed and can be synthesized using other operations.

Bit Set	The bit set operation allows you to set bit <i>i</i> of a word to 1.
Bit Clear	The bit clear operation allows you to clear bit <i>i</i> of a word to 0.
Bit Toggle	The bit toggle operation allows you to complement bit i of a word to its complement.

ARM Logical Operations

Few microprocessors implement all the above logical operations. Some microprocessors implement special-purpose logical operations as we shall see. The ARM's logical operations are:

MVN	MVN r0 , r1	$r0 = \overline{r1}$
AND	AND r0 , r1, r2	$r0 = r1 \cdot r2$
ORR	OR r0 , r1, r2	r0 = r1 + r2
EOR	XOR r0 , r1, r2	$r0 = r1 \oplus r2$
BIC	BIC r0 , r1, r2	$r0 = r1 \cdot r\overline{2}$
LSL	MOV r0 , r1, LSL r2	r1 is shifted left by the number of places in r2
LSR	MOV r0 , r1, LSR r2	r1 is shifted right by the number of places in r2

The two unusual instructions are MVN (move negated) and BIC (clear bits). The move negated instruction acts rather like a move instruction (MOV), except that the bits are inverted. Note that the bits in the source register remain unchanged. The BIC instruction clears bits of the first operands when bits of the destination operand are set. This operation is equivalent to an AND between the first and negated second operand. For example, in 8 bits the operation BIC r0, r1, r2 (with r1 = 00001111 and r2 = 11001010) would result in r0 = 11000000. This instruction is sometimes called *clear ones corresponding*.

The problem

Let's perform a simple Boolean operation to calculate the bitwise calculation of $F = A \cdot B + C \cdot D$. Assume that A, B, C, D are in r1, r2, r3, r4, respectively. The Code

AND r0 , r1, r2	;r0 = $A \cdot B$
AND r3 , r3, r4	$;r3 = C \cdot D$
MVN r3 ,r3	$r3 = \overline{C \cdot D}$
ORR r0 , r0, r3	$r0 = A \cdot B + \overline{C \cdot D}$

Figure Example 7.1 gives a snapshot of the state of this program after execution. There are several points of interest. First, we have used the pseudo instruction LDR r1, =2_00000001111111010101011110000 to load a binary literal into register r1.

The ARM cannot load a 32-bit constant in a single instruction (since an instruction itself is 32 bits wide). The LDR **r1**, = format generate a program counter relative load and the assembler automatically puts the constant in memory at a suitable point. Note the format of a base two number. It is $2 \times \times \times \times \times \times$ where the 2 indicates binary and the x's are bits.

Figure Example 7.1 The state of the system after executing the program



Example 8 A More Complex Logical Operation

Suppose we have three words P, Q and R. We are going to apply logical operations to subfields (bit fields) of these registers. We'll use 16-bit arithmetic for simplicity.

Suppose that we have three 6-bit bit fields in Q, R, and R as illustrated below. The bit fields are in red and are not in the same position in each word. A *bit field* is a consecutive sequence of bits that forms a logic entity. Often they are data fields packed in a register, or they may be graphical elements in a display (a row of pixels). However, the following example demonstrates the type of operation you may have to perform on bits.

```
 \begin{array}{ll} P = p_{15} \; \textbf{p_{14}} \; \textbf{p_{13}} \; \textbf{p_{12}} \; \textbf{p_{11}} \; \textbf{p_{10}} \; \textbf{p_{9}} \; \textbf{p_{8}} \; \textbf{p_{7}} \; \textbf{p_{6}} \; \textbf{p_{5}} \; \textbf{p_{4}} \; \textbf{p_{3}} \; \textbf{p_{2}} \; \textbf{p_{1}} \; \textbf{p_{0}} &= 0010000011110010 \\ Q = q_{15} \; q_{14} \; q_{13} \; q_{12} \; q_{11} \; q_{10} \; \textbf{q_{9}} \; \textbf{q_{8}} \; \textbf{q_{7}} \; \textbf{q_{6}} \; \textbf{q_{5}} \; \textbf{q_{4}} \; \textbf{q_{3}} \; \textbf{q_{2}} \; \textbf{q_{1}} \; \textbf{q_{0}} &= 0011000011110000 \\ R = r_{15} \; r_{14} \; r_{13} \; r_{12} \; r_{11} \; \textbf{r_{10}} \; \textbf{r_{9}} \; \textbf{r_{8}} \; \textbf{r_{7}} \; \textbf{r_{6}} \; \textbf{r_{5}} \; \textbf{r_{4}} \; \textbf{r_{3}} \; \textbf{r_{2}} \; r_{1} \; r_{0} &= 1100010011111000 \\ \end{array}
```

In this example we are going to calculate $F = (P + Q \oplus R) \cdot 111110$ using the three 6-bit bit fields.

Assuming that P, Q, and R are in registers r1, r2, and r3, respectively, we first have to isolate the required bit fields. Since we are going to assume that the original data is in memory, it doesn't matter if we modify these registers. In each case we use a move instruction and right shift the register by the number of placed required to right-justify the bit field.

right justify P;	#9	r1, r1, LSR	MOV
right justify Q	#1	r2, r2,LSR	MOV
right justify R;	#5	r3, r3,LSR	MOV

We now have

We also want to ensure that all the other bits of each register are zero. We can use a logical AND operation for this. Note that 0x3F is the 6-bit mask 111111. We could have used 2_111111

AND	r1, r1,#0x3F	;convert P to six significant bits right-justified
AND	r2, r2, # 0x3F	;do Q
AND	r3, r3 , #0x3F	;do R

The now leaves us with

$\mathbf{P} = 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \mathbf{p_{14}}\ \mathbf{p_{13}}\ \mathbf{p_{12}}\ \mathbf{p_{11}}\ \mathbf{p_{10}}\ \mathbf{p_{9}}$	= 000000000010000
$\mathbf{Q} = 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \mathbf{q_6}\ \mathbf{q_5}\ \mathbf{q_4}\ \mathbf{q_3}\ \mathbf{q_2}\ \mathbf{q_1}$	= 000000000111000
$\mathbf{R} = 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \mathbf{r_{10}}\ \mathbf{r_{9}}\ \mathbf{r_{8}}\ \mathbf{r_{7}}\ \mathbf{r_{6}}\ \mathbf{r_{5}}$	= 000000000100111

Now we can do the calculation.

EOR	r2, r2 , r3	;Calculate $Q \oplus R$
ORR	r2, r2,r1	;Logical OR with r1 to get $(P + Q \oplus R)$
AND	r3, r3 , #0x3E	;And with 111110 to get $(P + Q \oplus R) \cdot 111110$

Figure Example 8.1 gives a snapshot of the screen after we've run this program. Note the final code. After each operation, we put the result in a new register. You would not do this in practice; you would reuse registers. We've done this because you can see all intermediate results from the final snapshot of the program and that make debugging very easy.

Figure Example 8.1 The state of the system after executing the program

2 0 0 F	10-11 11	Pigsh Debug Pepiphenals Jobis 2VCS Window Help Pigsh Debug Pepiphenals Jobis Intervenals					
legisters	4	×	1] Examp	les Log	icalOps.s	2
Register	Volue		01		AREA	ExampleS, CODE, READONLY	6
Current		-61	02		ENTR	Y	
R0	0x00000000		03	1			
R1	0x00000000		100	1	LDB	r1,-2_0010000011110010	/set up P
-R2	0x00000000		05		LDR	r2,=2_0011000011110000	set up G
- R3	0x00000000		06		LDB	r3,=2_1100010011111000	7set up R
R4	0x00000000		07				
R5	0x00000000		06		MOV	r4, r1, LSR #9	sright justify P
R6	0x000000000		09		NOV	r5, r2, LSR #1	tright justify g
87	0x00000000		10		MON	r6, r3, LSB #5	tright justify R
R8	0x00000000		11				
R9	0x000000000		12		AND	r7, r4, #0x3F	sconvert P to six significant bits right-justifie
- R10	0x00000000		13		AND	r8, r5, #0x3F	J db C
R11	0x00000000	11	14		AND	r9, r6, #0x3F	sdo R
R12	0x00000000		15				
R13 (SP)	0x00000000		16		BOB	r10, r8, r9	/Celculate Q ZOR R
R14 (LR)	0x00000000		17		ORR	r11, r7, r10	/Logical CR evault with P to get (P + Q ECR R)
R15 (PC)	Cx00000000		18		AND	r12, r11, #0x3E	(AND with 111110 to get (F + 0 EOR R).111110
CP5R	0x00000003		19		12		
I ≤ SP5R	0x00000000		20	Toob	в	roob	:Stop here
H User/System			21		END		
Project Reg	isters						
							Transition list on the

We now demonstrate that you can simplify this code. We perform the same action but reuse registers. Moreover, we will do not bother truncating registers to six bits because we can perform that operation when we carry out the final AND with 1111110. You could not do this with numeric values, but you can with logical values because there is no carry out between columns. Figure Example 8.2 demonstrates that we get the same result.

```
AREA Example8, CODE, READONLY
ENTRY
                                   ; Calculate F = (P + Q \oplus R) \cdot 111110
LDR r1, =2 0010000011110010 ; Load P
LDR r2,=2 0011000011110000 ;Load Q
LDR r3, =2 1100010011111000 ; Load R
MOV r2, r2, LSR #1
                                   ; Right justify Q one place
EOR r2, r2, r1, LSR #5
                                   ; Calculate Q EOR R (and right justify R five places)
     r2, r2, r1, LSR #9
                                   ; Logical OR with P to get (P + Q EOR R)
ORR
                                   ; AND with 111110 to get (P + Q EOR R).111110
AND
      r2, r2, #0x3F
В
      Loop
                                   ; Stop here
END
```

Loop

```
T:\CengageEdition_2\StructuredARMexamples\Example8_LogicalOpsALT.uvproj - yv
                                                                                                                                                              file Edit View Project Figsh Debug Peripherals Jools SVCS Window Help
                                                                建運用用 10
                                                                                                           * 💷 🖉 🖉 🥥 🖉 🛄 • 🥆
     🐸 🖬 🕼
 왕 🗷 이 러마티 10 수 코 (2) 🗿 = (2) 🖓 · 🏾 · (2) · 🖉 · 🖉 · 🖉 · 🖉 · 🖉 · 🖉 ·
Registers
                                ŧ×
                                          Example8_LogicalOpsALT.s
                                                                                                                                                                 * X
 Register
                                 .
                                                     AREA Examples ALT, CODE, READONLY
                                                                                                                                                                    1X
                    Value
                                         03
                                         02
                                                     ENTRY
                                                                                            Calculate F = (P + Q EOR R).111110
     Current
                                                     LDB r1,-2 001000001110010 :Load P
LDB r2,-2 001100001110000 :Load Q
LDB r3,-2 110001001111000 :Load R
        RD
                     0x00000000
                                         03
                                         04
05
05
        R1
                     0x000020F2
       R2
                     0x0000001E
                                                     MOV
                                                           r2, r2, LSR #1
                     0x0000C4F8
                                                                                            Right justify Q one place
                                                           r2,r2,r3, LSR #1
r2,r2,r3, LSR #5
r2,r2,r1, LSR #9
                                                                                           :Calculate Q EOR R (and right justify R five places) ;
Logical OR with P to get (P + Q EOR R). Justify R ;
AND with 111110 to get (P + Q EOR R).111110
        R4
                     0x00000000
                                         07
08
                                                     EOR
                                                     ORR
        R5
                     0x00000000
                                         09
                                                     AND
                                                           r2, r2, #0x3E
        RS
                     0x00000000
        B7
                     0x00000000
                                         10
                                             Loop
                                                     E
                                                            Loop
                                                                                            ;Stop here
                                                     END
        Rŝ
                     0x00000000
                                         11
        R9
                     0x00000000
                                         12
        B10
                     0x00000000
                                         13
        R11
                     0x00000000
        R12
                     0x00000000
                     0x00000000
        R13 (SP)
        R14(LR)
                     0x00000000
       R15 (PC)
                     0.0000010
 Project | Registers
                                      4
                                                                                                                                                                 Simulation
                                                                                                                                                         t1: 0.000000
```

Figure Example 8.2 The state of the system after executing the simplified program

Example 9 Conditional Expressions TO BE COMPLETED