FAQ on ARM assembly language programming, ver.2b

1. Question

What is DCD and how to use it?

Answer

Answer:

 DCB Reserve a 8-bit value

 DCW Reserve a 16-bit value

 DCD Reserve a word (32 bit value)

 DCS Reserve a string of 255 bytes

e.g

valuex DCD 0,1,5,4,0xffffeeee; reserved five 32-bit memory locations with 5 initialized values 0,1,5,4,0xffffeeee. The symbolic name of the first location is valuex.

1. Question on the use of directives to define the program

How to read the beginning of the following typical ARM program?

1) AREA |.data|, DATA, READWRITE; define RAMspace from0x40000000

2)value1 DCD 0, 0 ; declare two 32-bit memory space

3) align ; align the following address at a 4-byte boundary

4);; so the address here is at a 4 byte boundary

5)

6); User Initial Stack & Heap ;if needed you may your heap/stack space

7) AREA |.text|, CODE, READONLY ; define program from 0x0

8)

9) EXPORT \_\_main ; if a c-prohram calls this .s,it is the entry

10)\_\_main nop;BL iuart0

11) ;BL idata ;

12) mov r7,#0x02 ; uinit to 0x02, mov r7,0x02 is wrong

Answer:

Line1) Define working RAM memory space, lpc2131 is from 0x40000000.

Line2) Use “DCD 0,0”, to declare a space of two integer (32-bit each) variables , the symbolic name of the first is “value1” at address “=value1”. They are initialized as 0 and 0.

Line3) “align” is added to make the address following this to be at a 4-byte boundary (the address dividable by 4)

Line6) you may add directives to reserve RAM space for heap and stack

Line7) program starts from here, in arm-lpc2131, it is from address 0x0.

Line9) EXPORT \_\_main ;if a c-program calls this .s,it is the entry

1. Question on the use of LDR,ADR

Are there any difference between statement 1,2,3 in the following program?

Data1p DCD 0, 0 ;just happen the address is at 0x40000000

;DCD (reserve a 32-bit word)is a pseudo instruction to

;allocate a memory location for this data.

 align

 align

:

:

Statment1 LDR R0, =Data1p ; Put the address Data1p into R0

Statment2 ADR R0, Data1p ; Put the address Data1p into R0

Statment3 LDR R0, =0x40000000 ; Put the value0x40000000 into R0,

 ;just happen it is the address of Data1p

Answer: They are all the same, every one generates the same result. Such that the address ( not the data content) will be saved into R0. As a result the value in R0 will become 0x40000000.

1. Question on MOV

Does the instruction “Mov” have indirect addressing?

Answer. No, e.g. you cannot use mov r1,[r2]

*“MOV loads a value into the destination register, from another register, a shifted register, or an immediate 8-bit value.”*

Examples:

 MOV R0, R1 if R1 has 0x00001234, after this ,R1=R2=0x00001234

 MOV R0, #0x12; after this R0 has #0x12

 MOV R0, #300; is wrong the value should be less than 255

 MOV R0, 200; is wrong, # is missing “mov R0,#200: is correct

Note: the immediate value must be prefixed by #

1. Question on LDR/STR

(a)

How to store a data from R3 into a memory address location?

Answer: we will use indirect addressing, in which, R0 is used as the pointer.

Example

Data1p DCD 0, 0 ; the data is in this address Data1p,

;assume it is at address location 0x43210000

;DCD (reserve a 32-bit word)is a pseudo instruction to

;allocate a memory location for this data.

 align

:

:

 MOV R3, 0x00000003 ;place 0x00000003 into R3

 LDR R0, =Data1p ;save the address (pointer) of the data into R0

 STR R3, [R0]; uses R0 as the address (pointer, store content

;of R3 into the memory address location which should be 0x43210000

;after that, location 0x43210000 should contain 0x00000003

(b)

How to load a data from a memory address into R3?

Answer:

 LDR R0, =Data1p ;save the address (pointer) of the data into R0

 LDR R3, [R0]; use R0 as the address (pointer) and

;save the data in R3, after this, R3 should contain 0x00000003

And how to store a data from R3 to a memory address?

1. Question on differences between LDR and MOV

What is the difference between MOV and LDR, they seem to have the same function of saving information into registers?

Answer part 1: How different are they?

Note: “#” for mov, “=” for ldr. To define an immediate value

* + MOV can only move an 8-bit value (0x00->0xff=255) into a register while LDR can move a 32-bit value into a register. The immediate value is prefixed by different characters in mov and ldr: “#” for mov, “=” for ldr. E.g.

Mov r1,#255 ; ok, 255 is the biggest number you can mov

Mov r1,255 ; is wrong , missing #

Mov r1,#256 ; is wrong, the number is bigger than 255

Mov r1,#0x12340000 ; is wrong, the number is bigger than 255

Ldr r1,=255; you can do this,

Ldr r1,=256; you can do this,

Ldr r1,=0x12340000; you can do this,

* + MOV can run faster than LDR.
	+ LDR can move a data from a memory address to a register, MOV can only i) move data between two registers or ii) save a 8-bit immediate value to a register. e.g.

 value1 DCD 0; this define an integer variable “value1” with address “=value1”

:

;A standard pair of statements for moving a data into a register

Ldr r0,=value1 ; 1) save the address of the variable value1 in r0

Ldr r1,[r0] ;2)use r0 as the address (pointer) to get value1 to r1

Note: Mov cannot be used to achieve the same result, because mov r1,[r0] is not allowed

Answer part 2 : How similar are they?.

MOV is a real instruction (a 32-bit instruction) , LDR is a pseudo instruction (the assembler will use multiple 32-bit instructions to achieve the goal). For data move, if the immediate value is small, the assembler will use “mov” to implement it, so mov and ldr are exactly the same if the immediate value less or equal to 255. For example, for ldr r0,=14; the immediate value is 14 and is <255, so it will be implemented using mov r0,#14 (see the use of # and = ).

However, for a large immediate value (>255) “mov” does NOT work, e.g. ldr r0,=0x55555555; it is not ok to use mov r0,#0x55555555 because it is not allowed. Then, the assembler will generate some code to place the constant 0x55555555 in a nearby table in the code area. Then it uses an instruction to load a data from that table pointed by the program counter and an offset to fill up r0. The reason is because there is no way to fit a 32-bit data into a 32-instruction (an instruction must have the instruction-code-part and the data-part, if the data-part is 32-bit long, there is no room to store the instruction-code-part). Details can be found at <http://www.keil.com/support/man/docs/armasm/armasm_chdcegci.htm>

see also the directive “LTORG” for how to setup the table. Usually it is placed at the near by code area. You don’t need to worry too much because everything is automatic; you only need to place “LTORG” at the end of the code area as shown in the example at Keil.