GNU ARM Assembler Quick Reference

A summary of useful commands and expressions for the ARM architecture using the GNU assembler is presented briefly in the concluding portion of this Appendix. Each assembly line has the following format:

```
[<label>:]  [<instruction or directive>}  @ comment
```

Unlike the ARM assembler, using the GNU assembler does not require you to indent instructions and directives. Labels are recognized by the following colon instead of their position at the start of a line. An example follows showing a simple assembly program defining a function 'add' that returns the sum of two input arguments:

```
.section .text, "x"

.global     add                 @ give the symbol add external linkage

add:
      ADD   r0, r0, r1          @ add input arguments
      MOV   pc, lr              @ return from subroutine

                                @ end of program
```

GNU Assembler Directives for ARM

The follow is an alphabetical listing of the more command GNU assembler directives.

GNU Assembler Directive	Description
.ascii "<string>"	Inserts the string as data into the assembly (like DCB in armasm).
.asciz "<string>"	Like .ascii, but follows the string with a zero byte.
.balign <power_of_2> {,<fill_value> {,<max_padding>} }	Aligns the address to <power_of_2> bytes. The assembler aligns by adding bytes of value <fill_value> or a suitable default. The alignment will not occur if more than <max_padding> fill bytes are required (similar to ALIGN in armasm).
.byte <byte1> {,<byte2>} …	Inserts a list of byte values as data into the assembly (like DCB in armasm).
.code <number_of_bits>	Sets the instruction width in bits. Use 16 for Thumb and 32 for ARM assembly (similar to CODE16 and CODE32 in armasm).
.else	Use with .if and .endif (similar to ELSE in armasm).
.end	Marks the end of the assembly file (usually omitted).
.endif	Ends a conditional compilation code block – see .if, .ifdef, .ifndef (similar to ENDIF in armasm).
.endm	Ends a macro definition – see .macro (similar to MEND in armasm).
.endr	Ends a repeat loop – see .rept and .irp (similar to WEND in armasm).
.equ <symbol name>, <value>	This directive sets the value of a symbol (similar to EQU in armasm)
.err	Causes assembly to halt with an error.
.exitm	Exit a macro partway through – see .macro (similar to MEXIT in armasm)
.global <symbol>	This directive gives the symbol external linkage (similar to EXPORT in armasm).
.hword <short1> {,<short2>} …	Inserts a list of 16-bit values as data into the assembly (similar to DCW in armasm).

GNU Assembler Directive	Description
`.if <logical_expression>`	Makes a block of code conditional. End the block using .endif (similar to IF in armasm). See also .else.
`.ifdef <symbol>`	Include a block of code if <symbol> is defined. End the block with .endif.
`.ifndef <symbol>`	Include a block of code if <symbol> is not defined. End the block with .endif.
`.include "<filename>"`	Includes the indicated source file (similar to INCLUDE in armasm or #include in C).
`.irp <param> {,<val_1>} {,<val_2>} …`	Repeats a block of code, once for each value in the value list. Mark the end of the block using a .endr directive. In the repeated code block, use \<param> to substitute the associated value in the value list.

`.macro <name> {<arg_1} {,<arg_2>} … {,<arg_N>}`	Defines an assembler macro called <name> with N parameters. The macro definition must end with .endm. To escape from the macro at an earlier point, use .exitm. These directives are similar to MACRO, MEND, and MEXIT in armasm. You must precede the dummy macro parameters by \. For example:
	
	`.macro SHIFTLEFT a, b`
	`  .if \b < 0`
	`  MOV \a, \a, ASR #-\b`
	`  .exitm`
	`  .endif`
	`  MOV \a, \a, LSL #\b`
	`.endm`

`.rept <number_of_times>`	Repeats a block of code the given number of times. End with .endr.
`<register_name> .req <register_name>`	This directive names a register. It is similar to the RN directive in armasm except that you must supply a name rather than a number on the right (e.g., `acc .req r0`).
`.section <section_name> {,"<flags>"}`	Starts a new code or data section. Sections in GNU are called `.text`, a code section, `.data`, an initialized data section, and `.bss`, an uninitialized data section. These sections have default flags, and the linker understands the default names (similar directive to the armasm directive AREA). The following are allowable .section flags for ELF format files: <Flag> Meaning a allowable section w writable section x executable section
`.set <variable_name>, <variable_value>`	This directive sets the value of a variable. It is similar to SETA in armasm.
`.space <number_of_bytes> {,<fill_byte>}`	Reserves the given number of bytes. The bytes are filled with zero or <fill_byte> if specified (similar to SPACE in armasm).
`.word <word1> {,<word2>} …`	Inserts a list of 32-bit word values as data into the assembly (similar to DCD in armasm).

Assembler Special Characters / Syntax

Inline comment char: `'@'`
Line comment char: `'#'`
Statement separator: `';'`
Immediate operand prefix: `'#' or '$'`

Register Names

General registers:	`%r0 - %r15`	`($0 = const 0)`
FP registers:	`%f0 - %f7`	
Non-saved (temp) regs:	`%r0 - %r3, %r12`	
Saved registers:	`%r4 - %r10`	
Stack ptr register:	`%sp`	
Frame ptr register:	`%fp`	
Link (retn) register:	`%lr`	
Program counter:	`%ip`	
Status register:	`$psw`	
Status register flags:	`xPSR`	
(x = C current)	`xPSR_all`	
(x = S saved)	`xPSR_f`	
	`xPSR_x`	
	`xPSR_ctl`	
	`xPSR_fs`	
	`xPSR_fx`	
	`xPSR_fc`	
	`xPSR_cs`	
	`xPSR_cf`	
	`xPSR_cx`	
	`.. and so on`	

Arm Procedure Call Standard (APCS) Conventions

Argument registers:	`%a0 - %a4`	`(aliased to %r0 - %r4)`
Returned value regs:	`%v1 - %v6`	`(aliased to %r4 - %r9)`

Addressing Modes

`'rn'` in the following refers to any of the numbered registers, but not the control registers.

`addr`	Absolute addressing mode
`%rn`	Register direct
`[%rn]`	Register indirect or indexed
`[%rn,#n]`	Register based with offset
`#imm`	Immediate data

Machine Dependent Directives

`.arm`	Assemble using arm mode
`.thumb`	Assemble using thumb mode
`.code16`	Assemble using thumb mode
`.code32`	Assemble using arm mode
`.force_thumb`	Force thumb mode (even if not supported)
`.thumb_func`	Mark entry point as thumb coded (force bx entry)
`.ltorg`	Start a new literal pool

Opcodes

For detailed information on the machine instruction set, see this manual:

ARM Architecture Reference Manual, Addison-Wesley ISBN 0-201-73719-1

Here is a recommended book to get a lot of system developer information on the ARM architecture.

ARM System Developer's Guide, Morgan Kaufmann Publishers ISBN 1-55860-874-5 (alk.paper), authors: Andrew N. Sloss, Dominic Symes, Chris Wright, 2004