```
\ ARM Assembler.
\
\ This program is distributed under the terms of the 'MIT license'. The text
\ of this licence follows...
\
\ Copyright (c) 2005 J.D.Medhurst (a.k.a. Tixy)
\
\ Permission is hereby granted, free of charge, to any person obtaining a copy
\ of this software and associated documentation files (the "Software"), to deal
\ in the Software without restriction, including without limitation the rights
\ to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
\ copies of the Software, and to permit persons to whom the Software is
\ furnished to do so, subject to the following conditions:
\
\ The above copyright notice and this permission notice shall be included in
\ all copies or substantial portions of the Software.
\
\ THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
\ IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
\ FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.  IN NO EVENT SHALL THE
\ AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
\ LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
\ OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
\ THE SOFTWARE.
\
\
\ ------------------------------------------------------------------------
\ REQUIREMENTS and DEPENDECIES
\
\ This code requires ANS wordsets: CORE, CORE-EXT, SEARCH-ORDER and
\ SEARCH-ORDER-EXT. It also uses the non-standard word VOCABULARY.
\
\ The code is dependent on the size of a CELL being at least 32 bits.
\
\
\ ------------------------------------------------------------------------
\ USAGE
\
\ All assembler words are in the vocabulary ARM-ASSEMBLER. Before use
\ this must be added to the search order, with:
\
\     ALSO ARM-ASSEMBLER
\
\ Before performing any assembly CODE-BEGIN must be executed, this takes as
\ a parameter the target address for which code will be compiled for.
\ When all code has been assembled, CODE-END must be executed to ensure that
\ the final instruction is assembled correctly.
\
\ Because many ARM instructions also share names with standard Forth words
\ there may be problems when intermingling Forth and ARM code. To resolve this
\ the word [[ is provided which adds the FORTH wordlist to the beginning of
\ the search order, the word ]] reverts this change. E.g.
\
\     mov r0 [[ SOME-VARIABLE @ FF AND ]] #
\
\ This is analogous to escaping from compilation mode with the Forth [ ]
\ words.
\
\
\ ------------------------------------------------------------------------
\ ASSEMBLER SYNTAX
\
\ The assembler syntax attempts to follow that defined by ARM but changes have
\ made to enable it to work under the Forth interpreter. These changes are
\ detailed below.
\
\ 1. Operands are separated by whitespace rather than commas. E.g.
\
\        ADD  R0,R1,R2
\
\    is written as
```

```
\
\        ADD  R0 R1 R2
\
\ 2. Non alpha-numeric symbols must be surrounded by whitespace. E.g.
\
\        LDR  R0,[R1]
\        LDM  R0!,{R2,R3}
\
\    is written as
\
\        LDR  R0 [ R1 ]
\        LDM  R0 ! { R2 R3 }
\
\ 3. Condition code mnemonics must be surrounded by whitespace. E.g.
\
\        MOVEQ  R0,R1
\
\    is written as
\
\        MOV EQ  R0 R1
\
\    This may also mean that an instruction's name gets split. E.g.
\
\        LDREQBT  R0,[R1]
\        MOVEQS   R2,R0
\
\    is written as
\
\        LDR EQ BT  R0 [ R1 ]
\        MOV EQ S   R2 R0
\
\ 4. All numeric constants are postfixed by a #, (which actually just takes a
\    value from the stack). E.g.
\
\        MOV  R0,#123
\
\    is written as
\
\        MOV  R0 123 #
\
\    A # is also used for numeric values which don't normally have a # prefix
\    in ARM syntax. E.g.
\
\        MCR  P15,1,R2,C3,C4,5
\
\    is written as
\
\        MCR  P15 1 # R2 C3 C4 5 #
\
\ 5. The PSR fields in the MSR instruction must have whitespace before them.
\    E.g.
\
\        MSR  CPSR_CF,R0
\
\    is written as
\
\        MSR  CPSR_ CF R0
\
\    The words CPSR_ and SPSR_ are parsing words.
\
\
\ Other Notes:
\
\ The load an store instructions can take numeric constants as their
\ addressing mode and the assembler converts these into pc relative
\ addressing. E.g.
\
\     LDR R0 12345678 #
\
\ is converted to the form
\
```

```
\     LDR R0 [ PC <offset> # ]
\
\ The pseudo instruction ADR does a similar thing to generate ADD or SUB
\ instructions which give a pc relative address. E.g.
\
\     ADR R0 12345678 #
\
\ is converted to the form
\
\     ADD R0 PC <offset> #
\
\
\ LABELS
\
\ The assembler supports labels. There are 3 words which support this:
\
\    L: name
\        Define label 'name' whose value is the current code address.
\
\    L= name
\        Define label 'name' and assign its value from the top of the stack.
\
\    L# name
\        Use the value of label 'name' in an instruction as though it were a
\        numeric constant.
\
\    Examples:
\        CHAR " L= terminator
\        L: scan-loop
\            ldr b  r0 [ r1 ] 1 #
\            cmp    r0 L# terminator
\            b ne   L# scan-loop
\
\            ldr    r0 L# default
\            mov    pc lr
\        L: default
\            dcd 12345678 #
\
\
\ PORTING
\
\ The internal words CODE-HERE CODE-HERE! and CODE, are used for storing
\ assembled code into memory. This implementation uses the Forth dictionary
\ but these words can be modified to use some other location.
\
\ CODE, will also need porting if this code is run on a Forth system where
\ a cell isn't 32 bits.
\
\
\ ------------------------------------------------------------------------
\ CHANGES
\
\ 2006-11-09
\     Fixed some ANS Forth Standard compliancey issues.
\     * Made all hex values use upper-case letters.
\     * Fixed use of PRIVATE and PUBLIC so the compilation wordlist isn't
\       changed during compilation of a word.
\     Thanks to Dmitry Yakimov for fixes and suggestions.
\
\
\ Code follows...


HEX

\ ------------------------------------------------------------------------
\ Place assembler in its own vocabulary.
\ Internal implementation words go into a separate PRIVATE-WORDLIST

VOCABULARY ARM-ASSEMBLER      ALSO ARM-ASSEMBLER DEFINITIONS
VOCABULARY PRIVATE-WORDLIST   ALSO PRIVATE-WORDLIST DEFINITIONS
```

```
ALSO FORTH   \ Make sure FORTH words are found first

: PRIVATE   ( -- )   \ Make new words go into private wordlist
        ALSO PRIVATE-WORDLIST DEFINITIONS PREVIOUS ;

: PUBLIC   ( -- )   \ Make new words go into public wordlist
        ALSO ARM-ASSEMBLER DEFINITIONS PREVIOUS ;

\ ------------------------------------------------------------------------
\ Helpers words...

: RROTATE   ( x1 u -- x2 )   \ Rotate the bits of x1 right by u bits
        1F AND
        2DUP RSHIFT
        ROT ROT 20 SWAP - LSHIFT
        OR
;

\ ------------------------------------------------------------------------
\ Words governing where assembled code gets placed.  ( PORTING )

: CODE-HERE   ( -- a-addr )   \ Return address where code will be assembled to
        HERE ;

: CODE-HERE!   ( a-addr -- )   \ Set address where code will be assembled to
        HERE - ALLOT ;

: CODE,   ( x -- )   \ Store 32 bit x at assembly address, and step to next
        , ;

\ ------------------------------------------------------------------------
\ Variables

VARIABLE OP-VALUE     \ bits for op-code being constructed
VARIABLE OP-MASK      \ mask of bits currently stored in OP-VALUE
VARIABLE OP-DEFAULT   \ mask of bits in OP-VALUE which have default values
VARIABLE SHIFT-FLAG   \ set by shift operands
VARIABLE ]-FLAG       \ set true by ]
VARIABLE {-FLAG       \ set true by {
VARIABLE REGISTER-LOCATION   \ 0 to 4 bytes giving location to store registers
VARIABLE '#           \ vector for #
VARIABLE 'RM          \ vector called when Rm register is encountered
VARIABLE CODE-ORIGIN  \ address at which OP-VALUE is assembled for

\ ------------------------------------------------------------------------
\ Words controlling immediate operands

PUBLIC

: #   ( x -- )   \ ARM Assembler, immediate operand suffix
        '# @ EXECUTE ;

PRIVATE

: UNEXPECTED-#   ( -- )
        TRUE ABORT" ARM Assembler: Unexpected #" ;

: RESET-#   ( -- )   \ Clear behaviour of immediate operand
        ['] UNEXPECTED-# '# ! ;

: INVALID#   ( -- )
        TRUE ABORT" ARM Assembler: Invalid immediate operand" ;

: CHECK#   ( u1 u2 -- u1 )   \ Check u1 <= u2
        OVER U< IF INVALID# THEN ;

\ ------------------------------------------------------------------------
\ Words for initialising op-code generation

: DEFAULT-RM   ( x1 x2 x3 -- x1 x2 )   \ Default action when RM register used
```

```
        DROP ;   \ Do nothing

: OP-RESET   ( -- )   \ Reset assembler state
        0 OP-VALUE !
        0 OP-MASK !
        F0000000 OP-DEFAULT !
        0 SHIFT-FLAG !
        0 ]-FLAG !
        0 {-FLAG !
        FFFFFFFF REGISTER-LOCATION !
        RESET-#
        ['] DEFAULT-RM 'RM !
;

: ?INVALID   ( x -- )   \ If x not zero, then instruction is invalid
        ABORT" ARM Assembler: Invalid instruction form"
;

: OP-BUILD   ( x1 x2 -- )   \ Set opcode bits masked by x2 to values in x1
        OP-MASK @
        2DUP AND ?INVALID
        OVER OR OP-MASK !
        INVERT OP-VALUE @ AND OR OP-VALUE !
;

: OP-END   ( -- )   \ End of instruction assembly
        OP-MASK @
        IF
                OP-MASK @ OP-DEFAULT @ OR
                FFFFFFFF XOR ?INVALID   \ check all bits are accounted for
                OP-VALUE @ CODE,
                4 CODE-ORIGIN +!
                0 OP-MASK !
        THEN
;

: OP-INIT   ( x1 x2 x3 -- )   \ Initialise the assembler of new instruction
        OP-RESET
        REGISTER-LOCATION !
        OP-MASK !
        OP-VALUE !
;

: OP-BEGIN   ( x1 x2 x3 -- )    \ Start assembly of a new instruction
        OP-END OP-INIT ;

: DO-INSTRUCTION   ( a-addr -- )   \ Common behaviour of INSTRUCTION words
        \ Fetch 3 words from a-addr and call OP-BEGIN
        DUP @
        SWAP CELL+ DUP @
        SWAP CELL+ @
        OP-BEGIN
;

: INSTRUCTION   ( "<spaces>name" -- )   \ Create instruction
        CREATE
        DOES>  ( -- )
        DO-INSTRUCTION
;

: INSTRUCTION#   ( "<spaces>name" -- )   \ Instruction with immediate operand
        CREATE
        DOES>  ( -- )
        DUP DO-INSTRUCTION
        CELL+ CELL+ CELL+
        @ '# !
;

: INSTRUCTIOND   ( "<spaces>name" -- )   \ Instruction with default ops mask
        CREATE
        DOES>  ( -- )
```

```
        DUP DO-INSTRUCTION
        CELL+ CELL+ CELL+
        @ OP-DEFAULT !
;

: INSTRUCTIOND#   ( "<spaces>name" -- )   \ Instr. with default and immediate
        CREATE
        DOES>  ( -- )
        DUP DO-INSTRUCTION
        CELL+ CELL+ CELL+
        DUP @ '# !
        CELL+ @ OP-DEFAULT !
;

\ ------------------------------------------------------------------------
\ Flags parsing

: LOWER-CASE   ( c1 -- c2 )   \ Covert ASCII character to lower-case
        DUP [CHAR] A - 1A U< 20 AND +
;

: FLAG   ( c c-addr -- )   \ Process a single parsed flag character
        BEGIN
                2DUP C@ <>
        WHILE
                DUP C@ 0= ?INVALID
                CHAR+ CHAR+
        REPEAT
        1 SWAP CHAR+ C@ LSHIFT
        DUP OP-BUILD
        DROP
;

: FLAGS   ( -- )   \ Create flags parsing word
        CREATE
        DOES>   ( "<spaces>ccc" -- )
        >R
        \ set flags in OP-VALUE for each flag char present in "ccc"...
        BL WORD COUNT
        BEGIN
                DUP
        WHILE
                OVER C@ LOWER-CASE R@ FLAG
                1- SWAP CHAR+ SWAP
        REPEAT
        2DROP
        \ now set OP-MASK for each flag bit...
        0 R>
        BEGIN
                DUP C@
        WHILE
                CHAR+
                >R 1 R@ C@ LSHIFT OR
                R> CHAR+
        REPEAT
        DROP
        OP-MASK @ OR OP-MASK !
;

\ ------------------------------------------------------------------------
\ Labels
\
\ Structure of object is
\   CELL    REF-LINK   \ link to list of unresolved references
\   CELL    VALUE      \ value for label
\   STRING  NAME       \ name of label as a counted string

: LABEL>REF-LINK   ( a-addr1 -- a-addr2 )
        ;

: LABEL>VALUE   ( a-addr1 -- a-addr2 )
```

```
        CELL+ ;

: LABEL>NAME   ( a-addr -- c-addr )
        CELL+ CELL+ ;

0F CONSTANT MAX-LABEL-NAME-SIZE   \ Max significant length for label name

2 CELLS  MAX-LABEL-NAME-SIZE 1+ CHARS +   ALIGNED
CONSTANT LABEL-SIZE   \ Size of label object

10 CONSTANT #LABELS   \ Max number of labels
CREATE LABELS   #LABELS LABEL-SIZE * ALLOT
HERE CONSTANT LABELS-END

: ALLOC-LABEL   ( -- a-addr )   \ Allocate a new label object
        LABELS
        BEGIN
                DUP LABEL>NAME C@
        WHILE
                LABEL-SIZE +
                DUP LABELS-END U<
        WHILE
        REPEAT
                1 ABORT" ARM Assembler: Too many labels"
        THEN
;

: CREATE-LABEL   ( c-addr1 u1 -- a-addr )
        MAX-LABEL-NAME-SIZE MIN
        ALLOC-LABEL >R
        R@ LABEL>NAME
        2DUP C!
        CHAR+ SWAP CHARS MOVE
        R>
;

: EQUAL   ( c-addr1 u1 c-addr2 u2 -- flag )   \ flag true if strings same
        ROT OVER <>
        IF DROP 2DROP FALSE EXIT THEN
        0 ?DO
                OVER I + C@
                OVER I + C@
                <> IF UNLOOP 2DROP FALSE EXIT THEN
        LOOP
        2DROP TRUE
;

: FIND-LABEL   ( c-addr u -- a-addr true | c-addr u false )
        MAX-LABEL-NAME-SIZE MIN
        2>R
        LABELS
        BEGIN
                DUP LABEL>NAME COUNT
                2R@ EQUAL 0=
        WHILE
                LABEL-SIZE +
                DUP LABELS-END U<
        WHILE
        REPEAT
                DROP 2R> FALSE
                EXIT
        THEN
        2R> 2DROP TRUE
;

\ ------------------------------------------------------------------------
\ Unresolved Label references
\
\ Structure of object is
\   CELL REF-LINK   \ link to next unresolved references
\   CELL ORIGIN     \ value for OP-ORIGIN where reference occurred
```

```
\   CELL OP-ADDR    \ address of instruction where reference occurred
\   CELL '#         \ xt of word handling immediate arguments for reference

: REF>REF-LINK   ( a-addr1 -- a-addr2 )
        ;

: REF>ORIGIN   ( a-addr1 -- a-addr2 )
        CELL+ ;

: REF>OP-ADDR   ( a-addr1 -- a-addr2 )
        CELL+ CELL+ ;

: REF>'#   ( a-addr1 -- a-addr2 )
        CELL+ CELL+ CELL+ ;

4 CELLS CONSTANT REF-SIZE

20 CONSTANT #LABEL-REFS   \ Max number of unresolved label references
CREATE LABEL-REFS   \ Size of label reference object
#LABEL-REFS REF-SIZE * ALLOT
HERE CONSTANT LABEL-REFS-END

: ALLOC-LABEL-REF   ( -- a-addr )   \ Allocate a reference to a label
        LABEL-REFS
        BEGIN
                DUP REF>ORIGIN @
        WHILE
                REF-SIZE +
                DUP LABEL-REFS-END =
                IF 1 ABORT" ARM Assembler: Too many label references" THEN
        REPEAT
;

: CREATE-LABEL-REF   ( a-addr -- )   \ Create reference to label a-addr
        ALLOC-LABEL-REF >R

        DUP LABEL>REF-LINK @      \ get old head of ref list
        R@ REF>REF-LINK !         \ link to old head from new ref
        R@ SWAP LABEL>REF-LINK !  \ make new ref the head

        CODE-ORIGIN @ R@ REF>ORIGIN  !
        CODE-HERE R@ REF>OP-ADDR !
        '# @ R> REF>'# !
;

: RESOLVE-REF   ( x a-addr -- x a-addr )   \ Ref a-addr resolves to value x
        CODE-ORIGIN @ >R

        >R
        OP-RESET

        FFFFFF10 REGISTER-LOCATION !
        R@ REF>ORIGIN @ CODE-ORIGIN !
        R@ REF>OP-ADDR @ @ OP-VALUE !

        DUP R@ REF>'# @ EXECUTE

        OP-VALUE @
        R@ REF>OP-ADDR @ !

        OP-RESET
        R> 0 OVER REF>ORIGIN !   \ clear ref origin to indicate it is free

        R> CODE-ORIGIN !
;

: LABEL-RESOLVE   ( a-addr -- )   \ Resolve all references to a label
        DUP LABEL>VALUE @
        SWAP LABEL>REF-LINK
        BEGIN
                DUP @    \ get ref-link
```

```
                0 ROT !  \ clear ref-link
                DUP
        WHILE
                RESOLVE-REF
                REF>REF-LINK
        REPEAT
        2DROP
;

: CHECK-LABELS-RESOLVED   ( -- )
        0 LABELS
        BEGIN
                DUP LABEL>REF-LINK @
                IF
                        ." Unresolved label: " DUP LABEL>NAME COUNT TYPE CR
                        >R 1+ R>
                THEN
                LABEL-SIZE +
                DUP LABELS-END U< 0=
        UNTIL
        DROP
        ABORT" ARM Assembler: Unresolved labels"
;


\ -----------------------------------------------------------------------------
\ Top level words for labels

: LABELS-RESET   ( -- )   \ Clear all labels and references
        LABELS LABELS-END OVER - ERASE
        LABEL-REFS LABEL-REFS-END OVER - ERASE
;

: GET-LABEL   ( <spaces>"name" -- a-addr flag )   \ Find label or create one
        \ flag is true if label has a value assigned.
        BL WORD COUNT FIND-LABEL
        IF
                DUP LABEL>REF-LINK @ 0=
        ELSE
                CREATE-LABEL FALSE
        THEN
;

PUBLIC

: L=    ( x <spaces>"name" -- )   \ ARM Assembler, assign value to label
        OP-END
        GET-LABEL
        ABORT" Assembler: Label already defined"
        SWAP OVER LABEL>VALUE !
        LABEL-RESOLVE
;

: L:    ( <spaces>"name" -- )   \ ARM Assembler, define a label
        OP-END  CODE-ORIGIN @ L= ;

: L#    ( <spaces>"name" -- )   \ ARM Assembler, label reference
        GET-LABEL
        IF
                LABEL>VALUE @   \ use address from label
        ELSE
                CREATE-LABEL-REF
                CODE-ORIGIN @   \ use current address as a dummy value
        THEN
        '# @ EXECUTE
;

PRIVATE

\ -----------------------------------------------------------------------------
\ Pseudo instructions
```

```
PUBLIC

: .    ( -- addr )   \ ARM Assembler, current code address
        CODE-ORIGIN @ ;

PRIVATE

: DCD#   ( x -- )
        FFFFFFFF OP-BUILD ;

PUBLIC

INSTRUCTIOND# dcd    ( -- addr )   \ ARM Assembler, inline constant
        0 , 0 , FFFFFFFF , ' DCD# , 0 ,

PRIVATE

\ ------------------------------------------------------------------------
\ Top level words for assembler

PUBLIC

: [[   ( -- )   \ ARM Assembler, add FORTH wordlist to search order
        ALSO FORTH ;

: ]]   ( -- )   \ ARM Assembler, revert action of [[
        PREVIOUS ;

: CODE-BEGIN   ( a-addr -- )   \ Start code assembly
        CODE-ORIGIN !
        LABELS-RESET
        OP-RESET
;

: CODE-END   ( -- )   \ End code asembly
        OP-END
        CHECK-LABELS-RESOLVED
;

PRIVATE

\ ------------------------------------------------------------------------
\ Condition codes...

: CONDITION   ( u "<spaces>name" -- )   \ Definer for condition codes
        CREATE ,
        DOES>   ( -- )
        @ F0000000 OP-BUILD
;

PUBLIC

00000000 CONDITION eq   \ ARM Assembler, condition code
10000000 CONDITION ne   \ ARM Assembler, condition code
20000000 CONDITION cs   \ ARM Assembler, condition code
30000000 CONDITION cc   \ ARM Assembler, condition code
40000000 CONDITION mi   \ ARM Assembler, condition code
50000000 CONDITION pl   \ ARM Assembler, condition code
60000000 CONDITION vs   \ ARM Assembler, condition code
70000000 CONDITION vc   \ ARM Assembler, condition code
80000000 CONDITION hi   \ ARM Assembler, condition code
90000000 CONDITION ls   \ ARM Assembler, condition code
A0000000 CONDITION ge   \ ARM Assembler, condition code
B0000000 CONDITION lt   \ ARM Assembler, condition code
C0000000 CONDITION gt   \ ARM Assembler, condition code
D0000000 CONDITION le   \ ARM Assembler, condition code
E0000000 CONDITION al   \ ARM Assembler, condition code
F0000000 CONDITION nv   \ ARM Assembler, condition code

PRIVATE
```

```
\ ------------------------------------------------------------------------
\ Register operands

: ?BAD-REGISTER   ( x -- )
        ABORT" ARM Assembler: Unexpected or bad register operand"
;

: REGISTER   ( u "<spaces>name" -- )   \ Definer for CPU register words
        CREATE ,
        DOES>   ( -- )
        @ {-FLAG @
        IF
                \ register in ldm/stm list...
                DUP 0F U> ?BAD-REGISTER
                1 SWAP LSHIFT DUP OP-BUILD
                EXIT
        THEN

        \ get bit position for register in the op-code were building...
        REGISTER-LOCATION DUP @ DUP 8 RSHIFT FF000000 OR ROT !

        DUP -1 = ?BAD-REGISTER

        \ check register type
        2DUP XOR 20 AND ?BAD-REGISTER

        OVER >R

        \ create values for OP-BUILD
        SWAP 0F AND SWAP 1F AND
        0F OVER LSHIFT >R LSHIFT R>

        \ add op-code bits for shift value if required
        SHIFT-FLAG @
        IF
                SWAP 010 OR
                SWAP 090 OR
        THEN

        \ execute special action for RM register
        DUP 0F AND
        IF
                R@ 'RM @ EXECUTE
        THEN

        R> DROP
        OP-BUILD   \ include bits in op-code we're building
;

PUBLIC

00 REGISTER r0   \ ARM Assembler, CPU register
01 REGISTER r1   \ ARM Assembler, CPU register
02 REGISTER r2   \ ARM Assembler, CPU register
03 REGISTER r3   \ ARM Assembler, CPU register
04 REGISTER r4   \ ARM Assembler, CPU register
05 REGISTER r5   \ ARM Assembler, CPU register
06 REGISTER r6   \ ARM Assembler, CPU register
07 REGISTER r7   \ ARM Assembler, CPU register
08 REGISTER r8   \ ARM Assembler, CPU register
09 REGISTER r9   \ ARM Assembler, CPU register
0A REGISTER r10   \ ARM Assembler, CPU register
0B REGISTER r11   \ ARM Assembler, CPU register
0C REGISTER r12   \ ARM Assembler, CPU register
0D REGISTER r13   \ ARM Assembler, CPU register
0E REGISTER r14   \ ARM Assembler, CPU register
0F REGISTER r15   \ ARM Assembler, CPU register
80 REGISTER -r0   \ ARM Assembler, CPU register
81 REGISTER -r1   \ ARM Assembler, CPU register
82 REGISTER -r2   \ ARM Assembler, CPU register
```

```
83 REGISTER -r3   \ ARM Assembler, CPU register
84 REGISTER -r4   \ ARM Assembler, CPU register
85 REGISTER -r5   \ ARM Assembler, CPU register
86 REGISTER -r6   \ ARM Assembler, CPU register
87 REGISTER -r7   \ ARM Assembler, CPU register
88 REGISTER -r8   \ ARM Assembler, CPU register
89 REGISTER -r9   \ ARM Assembler, CPU register
8A REGISTER -r10   \ ARM Assembler, CPU register
8B REGISTER -r11   \ ARM Assembler, CPU register
8C REGISTER -r12   \ ARM Assembler, CPU register
8D REGISTER -r13   \ ARM Assembler, CPU register
8E REGISTER -r14   \ ARM Assembler, CPU register
8F REGISTER -r15   \ ARM Assembler, CPU register
20 REGISTER c0   \ ARM Assembler, coprocessor register
21 REGISTER c1   \ ARM Assembler, coprocessor register
22 REGISTER c2   \ ARM Assembler, coprocessor register
23 REGISTER c3   \ ARM Assembler, coprocessor register
24 REGISTER c4   \ ARM Assembler, coprocessor register
25 REGISTER c5   \ ARM Assembler, coprocessor register
26 REGISTER c6   \ ARM Assembler, coprocessor register
27 REGISTER c7   \ ARM Assembler, coprocessor register
28 REGISTER c8   \ ARM Assembler, coprocessor register
29 REGISTER c9   \ ARM Assembler, coprocessor register
2A REGISTER c10   \ ARM Assembler, coprocessor register
2B REGISTER c11   \ ARM Assembler, coprocessor register
2C REGISTER c12   \ ARM Assembler, coprocessor register
2D REGISTER c13   \ ARM Assembler, coprocessor register
2E REGISTER c14   \ ARM Assembler, coprocessor register
2F REGISTER c15   \ ARM Assembler, coprocessor register
: sp r13 ;     \ ARM Assembler, alias for r13
: -sp -r13 ;   \ ARM Assembler, alias for r13
: lr r14 ;     \ ARM Assembler, alias for r14
: -lr -r14 ;   \ ARM Assembler, alias for r14
: pc r15 ;     \ ARM Assembler, alias for r15
: -pc -r15 ;   \ ARM Assembler, alias for r15
PRIVATE

\ ------------------------------------------------------------------------
\ Shift operands

: SHIFT#   ( x -- )   \ Handle immediate operand for shifts
        DUP 1- SHIFT-FLAG @ U> IF INVALID# THEN
        1F AND 7 LSHIFT F90 OP-BUILD
        RESET-#
;

: SHIFT   ( n x -- )   \ Definer for shift operand words
        CREATE , ,
        DOES>   ( -- )
        DUP @ 060 OP-BUILD
        CELL+ @ SHIFT-FLAG !
        ['] SHIFT# '# !
        OP-DEFAULT @ FFFFF0FF AND OP-DEFAULT !   \ force more operands to be given
;

PUBLIC

1E 000 SHIFT lsl   ( -- )   \ ARM Assembler, shift operator
1F 020 SHIFT lsr   ( -- )   \ ARM Assembler, shift operator
1F 040 SHIFT asr   ( -- )   \ ARM Assembler, shift operator
1E 060 SHIFT ror   ( -- )   \ ARM Assembler, shift operator

: rrx   ( -- )   \ ARM Assembler, shift operator
        060 FF0 OP-BUILD ;

PRIVATE

\ ------------------------------------------------------------------------
\ Data processing instructions...

: ARM-DATA-LITERAL-ROTATE-COUNT   ( u -- u n )   \ Used by ARM-DATA-LITERAL
```

```
        0
        OVER 100 U<
        IF EXIT THEN
        BEGIN
                2 +
                2DUP RROTATE 0FF U>
        WHILE
                DUP 1E U<
        WHILE
        REPEAT
                DROP -1
        THEN
;

: ARM-DATA-LITERAL   ( u -- x )   \ Convert u into 12bit data literal
        ARM-DATA-LITERAL-ROTATE-COUNT
        DUP 0= IF DROP EXIT THEN
        DUP 20 U<
        IF
                DUP >R
                RROTATE
                20 R> - 7 LSHIFT OR
                EXIT
        THEN
        INVALID#
;

: DATA#   ( x -- )   \ Handle literal operand for data instructions
        ARM-DATA-LITERAL
        02000000 OR 02000FFF
        OP-BUILD
;

PUBLIC

INSTRUCTIOND# and   \ ARM Assembler, data processing instruction
        E0000000 , 0DE00000 , 0800100C , ' DATA# , F2100FFF ,

INSTRUCTIOND# eor   \ ARM Assembler, data processing instruction
        E0200000 , 0DE00000 , 0800100C , ' DATA# , F2100FFF ,

INSTRUCTIOND# sub   \ ARM Assembler, data processing instruction
        E0400000 , 0DE00000 , 0800100C , ' DATA# , F2100FFF ,

INSTRUCTIOND# rsb   \ ARM Assembler, data processing instruction
        E0600000 , 0DE00000 , 0800100C , ' DATA# , F2100FFF ,

INSTRUCTIOND# add   \ ARM Assembler, data processing instruction
        E0800000 , 0DE00000 , 0800100C , ' DATA# , F2100FFF ,

INSTRUCTIOND# adc   \ ARM Assembler, data processing instruction
        E0A00000 , 0DE00000 , 0800100C , ' DATA# , F2100FFF ,

INSTRUCTIOND# sbc   \ ARM Assembler, data processing instruction
        E0C00000 , 0DE00000 , 0800100C , ' DATA# , F2100FFF ,

INSTRUCTIOND# rsc   \ ARM Assembler, data processing instruction
        E0E00000 , 0DE00000 , 0800100C , ' DATA# , F2100FFF ,

INSTRUCTIOND# tst   \ ARM Assembler, data processing instruction
        E1100000 , 0DF0F000 , FF080010 , ' DATA# , F2000FFF ,

INSTRUCTIOND# teq   \ ARM Assembler, data processing instruction
        E1300000 , 0DF0F000 , FF080010 , ' DATA# , F2000FFF ,

INSTRUCTIOND# cmp   \ ARM Assembler, data processing instruction
        E1500000 , 0DF0F000 , FF080010 , ' DATA# , F2000FFF ,

INSTRUCTIOND# cmn   \ ARM Assembler, data processing instruction
        E1700000 , 0DF0F000 , FF080010 , ' DATA# , F2000FFF ,
```

```
INSTRUCTIOND# orr   \ ARM Assembler, data processing instruction
        E1800000 , 0DE00000 , 0800100C , ' DATA# , F2100FFF ,

INSTRUCTIOND# mov   \ ARM Assembler, data processing instruction
        E1A00000 , 0DEF0000 , FF08000C , ' DATA# , F2100FFF ,

INSTRUCTIOND# bic   \ ARM Assembler, data processing instruction
        E1C00000 , 0DE00000 , 0800100C , ' DATA# , F2100FFF ,

INSTRUCTIOND# mvn   \ ARM Assembler, data processing instruction
        E1E00000 , 0DEF0000 , FF08000C , ' DATA# , F2100FFF ,

: s   ( -- )   \ ARM Assembler, data processing instruction modifier
        00100000 DUP OP-BUILD
;

PRIVATE

\ ------------------------------------------------------------------------
\ Multiply instructions

PUBLIC

INSTRUCTIOND mul   ( -- )   \ ARM Assembler, multiply instruction
        E0000090 , 0FE0F0F0 , FF080010 , F0100000 ,
INSTRUCTIOND mla   ( -- )   \ ARM Assembler, multiply instruction
        E0200090 , 0FE000F0 , 0C080010 , F0100000 ,
INSTRUCTIOND umull   ( -- )   \ ARM Assembler, multiply instruction
        E0800090 , 0FE000F0 , 0800100C , F0100000 ,
INSTRUCTIOND umlal   ( -- )   \ ARM Assembler, multiply instruction
        E0A00090 , 0FE000F0 , 0800100C , F0100000 ,
INSTRUCTIOND smull   ( -- )   \ ARM Assembler, multiply instruction
        E0C00090 , 0FE000F0 , 0800100C , F0100000 ,
INSTRUCTIOND smlal   ( -- )   \ ARM Assembler, multiply instruction
        E0E00090 , 0FE000F0 , 0800100C , F0100000 ,

PRIVATE

\ ------------------------------------------------------------------------
\ Load/store instructions

: ADR#   ( x -- )   \ Handle immediate operand for ADR pseudo instructions
        CODE-ORIGIN @ 8 + -
        DUP 0<
        IF
                NEGATE
                OP-VALUE DUP @ 00C00000 XOR SWAP !   \ Turn add into sub
        THEN
        ARM-DATA-LITERAL
        00000FFF
        OP-BUILD
;

PUBLIC

INSTRUCTION# adr   ( -- )   \ ARM Assembler, pseudo op for PC relative address
        E28F0000 , 0FFF0000 , FFFFFF0C , ' ADR# ,

PRIVATE

: SET-PRE-INDEXING   ( x1 -- x2 )
        OP-MASK @ 01000000 AND 0=
        IF
                ]-FLAG @ 0=
                IF SWAP 01000000 OR SWAP THEN
                01000000 OR
        THEN
;

: TRY-MEM#   ( n -- n | n2 )   \ See if we need a PC relative address
        OP-MASK @ 000F000F AND 0=        \ if no Rn or index already set...
```

```
        IF pc CODE-ORIGIN @ 8 + - THEN   \ ...use pc relative addressing
;

: MEM#   ( n -- )   \ Process immediate operand for LDR/STR instructions
        TRY-MEM#                         \ check for absolute memory address
        DUP ABS 0FFF CHECK#              \ get abs value and check range
        SWAP 0< INVERT 00800000 AND OR   \ set U bit if index +ve
        02800FFF                         \ op-mask value
        SET-PRE-INDEXING                 \ set pre-index (P) bit if appropriate
        OP-BUILD
        RESET-#
;

: MEM-OP-RM   ( x1 x2 x3 -- x4 x5 )   \ Process Rm register for LDR/STR ops
        \ x1,x2 = op-value,op-mask   x3 = Rm register value
        \ x4,x5 = modified op-value,op-mask
        INVERT 80 AND 10 LSHIFT ROT OR   \ set U bit if not -ve register
        02000000 OR                      \ set register index
        SWAP 02800000 OR                 \ add bits to op-mask
        SET-PRE-INDEXING                 \ set pre-index (P) bit if appropriate
;

: MEM-OP   ( x -- )   \ Common factor for LDR and STR instructions
        0C100000 FF00100C OP-BEGIN
        ['] MEM# '# !
        F3E00FFF OP-DEFAULT !
        ['] MEM-OP-RM 'RM !
;

PUBLIC

: str   ( -- )   \ ARM Assembler, store instruction
        E5800000 MEM-OP ;

: ldr   ( -- )   \ ARM Assembler, load instruction
        E5900000 MEM-OP ;

: t   ( -- )   \ ARM Assembler, load and store instruction modifier
        00200000 01200000 OP-BUILD
;

: bt   ( -- )   \ ARM Assembler, load and store instruction modifier
        00600000 01600000 OP-BUILD
;

: [   ( -- )   \ ARM Assembler, address bracket
        ;

: ]   ( -- )   \ ARM Assembler, address bracket
        TRUE ]-FLAG ! ;

: !   ( -- )   \ ARM Assembler, address write-back modifier
        00200000 DUP OP-BUILD ;

PRIVATE

: EXTRA-MEM#   ( n -- )   \ Handle immediate operand for extra LDR/STR
        TRY-MEM#                              \ check for absolute memory address
        DUP ABS 0FF CHECK#                    \ get abs value and check range
        DUP 0F0 AND 4 LSHIFT SWAP 0F AND OR   \ covert value into op-code format
        00400000 OR                           \ set I bit
        SWAP 0< INVERT 00800000 AND OR        \ set U bit if index +ve
        00C00F0F                              \ op-mask value
        SET-PRE-INDEXING                      \ set pre-index bit if appropriate
        OP-BUILD
        RESET-#
;

: EXTRA-MEM-OP-RM   ( x1 x2 x3 -- x4 x5 )   \ Process Rm for extra LDR/STR
        \ x1,x2 = op-value,op-mask   x3 = Rm register value
        \ x4,x5 = modified op-value,op-mask
```

```
        INVERT 80 AND 10 LSHIFT ROT OR   \ set U bit if not -ve register
        SWAP 00C00000 OR                 \ add bits to op-mask
        SET-PRE-INDEXING                 \ set pre-index (P) bit if appropriate
;

: EXTRA-MEM-OP2   ( x -- )   \ Mutate into extra load/store instruction
        OP-VALUE @ F0000000 AND OR   \ add in condition code
        0E1000F0                     \ op-mask value
        OP-MASK @ F0000000 AND OR    \ add in condition code mask
        FF00100C OP-INIT             \ transmute instruction into new form
        F1E00F0F OP-DEFAULT !
        ['] EXTRA-MEM# '# !
        ['] EXTRA-MEM-OP-RM 'RM !
;

: EXTRA-MEM-OP   ( x -- )
        OP-VALUE @ 00100000 AND OR   \ add in ldr/str bit
        EXTRA-MEM-OP2
;

PUBLIC

: h   ( -- )   \ ARM Assembler, load and store instruction modifier
        01C000B0 EXTRA-MEM-OP ;

: sb   ( -- )   \ ARM Assembler, load and store instruction modifier
        01C000D0 EXTRA-MEM-OP ;

: sh   ( -- )   \ ARM Assembler, load and store instruction modifier
        01C000F0 EXTRA-MEM-OP ;

INSTRUCTIOND swp   ( -- )   \ ARM Assembler, swap instruction
        E1000090 , 0FB00FF0 , FF10000C , F0400000 ,

INSTRUCTIOND stm   ( -- )   \ ARM Assembler, store multiple instruction
        E8800000 , 0E100000 , FFFFFF10 , F1E0FFFF ,

INSTRUCTIOND ldm   ( -- )   \ ARM Assembler, load multiple instruction
        E8900000 , 0E100000 , FFFFFF10 , F1E0FFFF ,

PRIVATE

: MULTI-TYPE   ( x "<spaces>name" -- )
        CREATE ,
        DOES> ( -- )
        @ 01800000 OP-BUILD
;

PUBLIC

00000000 MULTI-TYPE da   \ ARM Assembler, load/store multiple modifier
00800000 MULTI-TYPE ia   \ ARM Assembler, load/store multiple modifier
01000000 MULTI-TYPE db   \ ARM Assembler, load/store multiple modifier
01800000 MULTI-TYPE ib   \ ARM Assembler, load/store multiple modifier

: {   ( -- )   \ ARM Assembler, load/store multiple register list brace
        TRUE {-FLAG ! ;

: }   ( -- )   \ ARM Assembler, load/store multiple register list brace
        ;

: ^ ( -- )   \ ARM Assembler, load/store multiple instruction modifier
        00400000 DUP OP-BUILD ;

PRIVATE

\ ------------------------------------------------------------------------
\ Branch instructions...

: ?BAD-BRANCH   ( flag -- )
        ABORT" ARM Assembler: Bad branch target"
```

```
;

: BRANCH#   ( x -- )   \ Handle immediate operand for branch instruction
        DUP 3 AND ?BAD-BRANCH   \ Check target is word aligned
        CODE-ORIGIN @ 8 + -
        DUP 0< OVER XOR FC000000 AND ?BAD-BRANCH   \ Check range for branch
        2 RSHIFT 00FFFFFF AND
        00FFFFFF OP-BUILD
        RESET-#
;

INSTRUCTION# BRANCH-OP   ( -- )
        EA000000 , 0F000000 , FFFFFFFF , ' BRANCH# ,

PUBLIC

: b   ( -- )   \ ARM Assembler, branch instruction or LDR/STR modifier
        OP-MASK @ DUP 0= SWAP 000FF000 AND OR
        IF BRANCH-OP EXIT THEN   \ branch instruction
        00400000 DUP OP-BUILD    \ byte modifier of load/store instructions
;

INSTRUCTION# bl   ( -- )   \ ARM Assembler, branch instruction
        EB000000 , 0F000000 , FFFFFFFF , ' BRANCH# ,

INSTRUCTION bx ( -- )   \ ARM Assembler, branch instruction (ARM 4T)
        E12FFF10 , 0FFFFFF0 , FFFFFF00 ,

PRIVATE

: SWI#   ( u -- )   \ Handle immediate operand for SWI instructions
        00FFFFFF CHECK#
        00FFFFFF OP-BUILD
        RESET-#
;

PUBLIC

INSTRUCTION# swi   ( -- )   \ ARM Assembler, swi instruction
        EF000000 , 0F000000 , FFFFFFFF , ' SWI# ,

PRIVATE

\ ------------------------------------------------------------------------
\ MRS and MSR instructions...

: PSR   ( x -- )   \ Common factor for PSR operands
        00400000 OP-BUILD
        OP-MASK @ 000F0000 AND IF EXIT THEN   \ end if mrs instruction
        00090000 000F0000 OP-BUILD   \ set f and c field mask
;

FLAGS PSR-FLAGS   \ Parse fields for PSR register
        CHAR f C, 13 C,
        CHAR s C, 12 C,
        CHAR x C, 11 C,
        CHAR c C, 10 C,
        0 C,

PUBLIC

: cpsr   ( -- )   \ ARM Assembler, status register
        00000000 PSR ;

: spsr   ( -- )   \ ARM Assembler, status register
        00400000 PSR ;

: cpsr_   ( -- )   \ ARM Assembler, status register
        00000000 00400000 OP-BUILD PSR-FLAGS ;

: spsr_   ( -- )   \ ARM Assembler, status register
```

```
        00400000 00400000 OP-BUILD PSR-FLAGS ;

INSTRUCTION mrs   ( -- )   \ ARM Assembler, status register instruction
        E10F0000 , 0FBF0FFF , FFFFFF0C ,

INSTRUCTIOND# msr   ( -- )   \ ARM Assembler, status register instruction
        E120F000 , 0DB0F000 , FFFFFF00 , ' DATA# , F2000FF0 ,

PRIVATE

\ ------------------------------------------------------------------------
\ Coprocessor instructions...

: COPROCESSOR    ( u "<spaces>name" -- )   \ Definer for coprocessor
        CREATE , DOES> @ 00000F00 OP-BUILD ;

PUBLIC

000 COPROCESSOR p0   \ ARM Assembler, Coprocessor
100 COPROCESSOR p1   \ ARM Assembler, Coprocessor
200 COPROCESSOR p2   \ ARM Assembler, Coprocessor
300 COPROCESSOR p3   \ ARM Assembler, Coprocessor
400 COPROCESSOR p4   \ ARM Assembler, Coprocessor
500 COPROCESSOR p5   \ ARM Assembler, Coprocessor
600 COPROCESSOR p6   \ ARM Assembler, Coprocessor
700 COPROCESSOR p7   \ ARM Assembler, Coprocessor
800 COPROCESSOR p8   \ ARM Assembler, Coprocessor
900 COPROCESSOR p9   \ ARM Assembler, Coprocessor
A00 COPROCESSOR p10   \ ARM Assembler, Coprocessor
B00 COPROCESSOR p11   \ ARM Assembler, Coprocessor
C00 COPROCESSOR p12   \ ARM Assembler, Coprocessor
D00 COPROCESSOR p13   \ ARM Assembler, Coprocessor
E00 COPROCESSOR p14   \ ARM Assembler, Coprocessor
F00 COPROCESSOR p15   \ ARM Assembler, Coprocessor

PRIVATE

: CO-MEM#   ( x -- )   \ Handle immediate operand for LDC/STC
        DUP
        {-FLAG @ 0= IF ABS 2 RROTATE THEN
        0FF CHECK#
        SWAP 0< INVERT IF 00800000 OR THEN
        008000FF
        ]-FLAG @
        IF
                \ Post index...
                {-FLAG @ 0= IF SWAP 00200000 OR SWAP THEN
                01200000 OR
        THEN
        OP-BUILD
        RESET-#
;

PUBLIC

INSTRUCTIOND# ldc   ( -- )   \ ARM Assembler, co-processor instruction
        ED900000 , 0E100000 , FFFF102C , ' CO-MEM# , F1E000FF ,

INSTRUCTIOND# stc   ( -- )   \ ARM Assembler, co-processor instruction
        ED800000 , 0E100000 , FFFF102C , ' CO-MEM# , F1E000FF ,

: l   ( -- )   \ ARM Assembler, co-processor instruction modifier
        00400000 DUP OP-BUILD ;

PRIVATE

: CO-OP2#   ( u -- )   \ Handle coprocessor instruction 2nd op-code
        7 CHECK#
        5 LSHIFT 000000E0 OP-BUILD
        RESET-#
;
```

```
: CO-DATA#   ( u -- )   \ Handler CDP instruction 1st op-code
        0F CHECK#
        14 LSHIFT 00F00000 OP-BUILD
        ['] CO-OP2# '# !
;

: CO-MOV#   ( u -- )   \ Handler MCR/MRC instruction 1st op-code
        7 CHECK#
        15 LSHIFT 00E00000 OP-BUILD
        ['] CO-OP2# '# !
;

PUBLIC

INSTRUCTION# cdp   ( -- )   \ ARM Assembler, co-processor instruction
        EE000000 , 0F000010 , FF20302C , ' CO-DATA# ,

INSTRUCTION# mcr   ( -- )   \ ARM Assembler, co-processor instruction
        EE000010 , 0F100010 , FF20300C , ' CO-MOV# ,

INSTRUCTION# mrc   ( -- )   \ ARM Assembler, co-processor instruction
        EE100010 , 0F100010 , FF20300C , ' CO-MOV# ,

PRIVATE

\ ------------------------------------------------------------------------
\ ARM5 instructions
\ ------------------------------------------------------------------------

: BLX#   ( x -- )   \ Handle immediate operand for BLX instruction
        0 F000000F OP-BUILD                    \ check no operands have been added
        DUP 2 AND 17 LSHIFT                    \ get half-word flag
        FA000000 OR FF000000 FFFFFFFF OP-INIT \ transmute instruction to long BLX
        FFFFFFFD AND                           \ clear half-word bit in address
        BRANCH#                                \ insert branch address
;

PUBLIC

INSTRUCTION# blx   ( -- )   \ ARM Assembler, branch instruction
        E12FFF30 , 0FFFFFF0 , FFFFFF00 , ' BLX# ,

PRIVATE

\ ------------------------------------------------------------------------

: BKPT#   ( u -- )   \ Handle immediate operand for BKPT instruction
        0000FFFF CHECK#
        DUP 0000FFF0 AND 4 LSHIFT
        SWAP 0000000F AND OR
        000FFF0F OP-BUILD
        RESET-#
;

PUBLIC

INSTRUCTION# bkpt   ( -- )   \ ARM Assembler, breakpoint instruction
        E1200070 , FFF000F0 , FFFFFFFF , ' BKPT# ,

PRIVATE

\ ------------------------------------------------------------------------

PUBLIC

INSTRUCTION clz   ( -- )   \ ARM Assembler, count leading zeros instruction
        E16F0F10 , 0FFF0FF0 , FFFF000C ,

PRIVATE
```

```
\ -----------------------------------------------------------------------------

PUBLIC

INSTRUCTION# mcr2   ( -- )   \ ARM Assembler, co-processor instruction
        FE000010 , FF100010 , FF20300C , ' CO-MOV# ,

INSTRUCTION# mrc2   ( -- )   \ ARM Assembler, co-processor instruction
        FE100010 , FF100010 , FF20300C , ' CO-MOV# ,

INSTRUCTIOND# ldc2   ( -- )   \ ARM Assembler, co-processor instruction
        FD900000 , FE100000 , FFFF102C , ' CO-MEM# , F1E000FF ,

INSTRUCTIOND# stc2   ( -- )   \ ARM Assembler, co-processor instruction
        FD800000 , FE100000 , FFFF102C , ' CO-MEM# , F1E000FF ,

INSTRUCTION# cdp2   ( -- )   \ ARM Assembler, co-processor instruction
        FE000000 , FF000010 , FF20302C , ' CO-DATA# ,

PRIVATE

\ -----------------------------------------------------------------------------
\ ARM5E instructions
\ -----------------------------------------------------------------------------

PUBLIC

: d   ( -- )   \ ARM Assembler, load/store double instruction modifier
        01C000D0
        OP-VALUE @ INVERT 00100000 AND 0F RSHIFT OR   \ add in ldr/str bit
        EXTRA-MEM-OP2
;

PRIVATE

\ -----------------------------------------------------------------------------

PUBLIC

: pld   ( -- )   \ ARM Assembler, PLD instruction
        ldr r15 F1400000 F1600000 OP-BUILD ;

PRIVATE

\ -----------------------------------------------------------------------------

: CO-MOV2#   ( u -- )   \ Handle immediate operand for MCRR/MRCC
        0F CHECK#
        4 LSHIFT 000000F0 OP-BUILD
        RESET-#
;

PUBLIC

INSTRUCTION# mcrr   ( -- )   \ ARM Assembler, co-processor double instruction
        EC400000 , 0FF00000 , FF20100C , ' CO-MOV2# ,

INSTRUCTION# mrrc   ( -- )   \ ARM Assembler, co-processor double instruction
        EC500000 , 0FF00000 , FF20100C , ' CO-MOV2# ,

PRIVATE

\ -----------------------------------------------------------------------------

PUBLIC

INSTRUCTION qadd   ( -- )   \ ARM Assembler, saturated arithmetic instruction
        E1000050 , 0FF00FF0 , FF10000C ,

INSTRUCTION qsub   ( -- )   \ ARM Assembler, saturated arithmetic instruction
        E1200050 , 0FF00FF0 , FF10000C ,
```

```
INSTRUCTION qdadd   ( -- )   \ ARM Assembler, saturated arithmetic instruction
        E1400050 , 0FF00FF0 , FF10000C ,

INSTRUCTION qdsub   ( -- )   \ ARM Assembler, saturated arithmetic instruction
        E1600050 , 0FF00FF0 , FF10000C ,

PRIVATE

\ ------------------------------------------------------------------------

PUBLIC

INSTRUCTION smlabb   ( -- )   \ ARM Assembler, dsp multiply instruction
        E1000080 , 0FF000F0 , 0C080010 ,

INSTRUCTION smlatb   ( -- )   \ ARM Assembler, dsp multiply instruction
        E10000A0 , 0FF000F0 , 0C080010 ,

INSTRUCTION smlabt   ( -- )   \ ARM Assembler, dsp multiply instruction
        E10000C0 , 0FF000F0 , 0C080010 ,

INSTRUCTION smlatt   ( -- )   \ ARM Assembler, dsp multiply instruction
        E10000E0 , 0FF000F0 , 0C080010 ,

INSTRUCTION smlawb   ( -- )   \ ARM Assembler, dsp multiply instruction
        E1200080 , 0FF000F0 , 0C080010 ,

INSTRUCTION smlawt   ( -- )   \ ARM Assembler, dsp multiply instruction
        E12000C0 , 0FF000F0 , 0C080010 ,

INSTRUCTION smulbb   ( -- )   \ ARM Assembler, dsp multiply instruction
        E1600080 , 0FF0F0F0 , FF080010 ,

INSTRUCTION smultb   ( -- )   \ ARM Assembler, dsp multiply instruction
        E16000A0 , 0FF0F0F0 , FF080010 ,

INSTRUCTION smulbt   ( -- )   \ ARM Assembler, dsp multiply instruction
        E16000C0 , 0FF0F0F0 , FF080010 ,

INSTRUCTION smultt   ( -- )   \ ARM Assembler, dsp multiply instruction
        E16000E0 , 0FF0F0F0 , FF080010 ,

INSTRUCTION smulwb   ( -- )   \ ARM Assembler, dsp multiply instruction
        E12000A0 , 0FF0F0F0 , FF080010 ,

INSTRUCTION smulwt   ( -- )   \ ARM Assembler, dsp multiply instruction
        E12000E0 , 0FF0F0F0 , FF080010 ,

INSTRUCTION smlalbb   ( -- )   \ ARM Assembler, dsp multiply instruction
        E1400080 , 0FF000F0 , 0800100C ,

INSTRUCTION smlaltb   ( -- )   \ ARM Assembler, dsp multiply instruction
        E14000A0 , 0FF000F0 , 0800100C ,

INSTRUCTION smlalbt   ( -- )   \ ARM Assembler, dsp multiply instruction
        E14000C0 , 0FF000F0 , 0800100C ,

INSTRUCTION smlaltt   ( -- )   \ ARM Assembler, dsp multiply instruction
        E14000E0 , 0FF000F0 , 0800100C ,

PRIVATE

\ ------------------------------------------------------------------------
\ ARM5J instructions
\ ------------------------------------------------------------------------

PUBLIC

INSTRUCTION bxj   ( -- )   \ ARM Assembler, branch instruction
        E12FFF20 , 0FFFFFF0 , FFFFFF00 ,
```

```
PRIVATE

\ ------------------------------------------------------------------------
\ ARM6 instructions
\ ------------------------------------------------------------------------

PUBLIC

INSTRUCTION# mcrr2   ( -- )   \ ARM Assembler, co-processor double instruction
        FC400000 , FFF00000 , FF20100C , ' CO-MOV2# ,

INSTRUCTION# mrrc2   ( -- )   \ ARM Assembler, co-processor double instruction
        FC500000 , FFF00000 , FF20100C , ' CO-MOV2# ,

PRIVATE

\ ------------------------------------------------------------------------

: CPS#   ( u -- )   \ Handle immediate operand for CPS instruction
        1F CHECK#
        00020000 OR 0002001F OP-BUILD
;

FLAGS IFLAGS   ( c -- )   \ Parse flags for CPS instruction
        CHAR a C, 8 C,
        CHAR i C, 7 C,
        CHAR f C, 6 C,
        0 C,

INSTRUCTIOND# cpsie   ( -- )   \ ARM Assembler, CPS instruction
        F1080000 , FFFDFE20 , FFFFFFFF , ' CPS# , F002001F ,

INSTRUCTIOND# cpsid   ( -- )   \ ARM Assembler, CPS instruction
        F10C0000 , FFFDFE20 , FFFFFFFF , ' CPS# , F002001F ,

PUBLIC

: cpsie   ( -- )
        cpsie IFLAGS ;

: cpsid   ( -- )
        cpsid IFLAGS ;

INSTRUCTION# cps   ( -- )   \ ARM Assembler, CPS instruction
        F1000000 , FFFDFFE0 , FFFFFFFF , ' CPS# ,

PRIVATE

\ ------------------------------------------------------------------------

PUBLIC

INSTRUCTION setend   ( -- )   \ ARM Assembler, SETEND instruction
        F1010000 , FFFFFDFF , FFFFFFFF ,

: le   ( -- )   \ ARM Assembler, modifier for SETEND instruction
        OP-VALUE @ F1010000 =   \ if SETEND instruction...
        IF 000 200 OP-BUILD     \ ... set LE bit to zero
        ELSE le                 \ else do less-or-equal condition code
        THEN
        ;

: be   ( -- )   \ ARM Assembler, modifier for SETEND instruction
        200 200 OP-BUILD ;

PRIVATE

\ ------------------------------------------------------------------------

: SRS#   ( u -- )   \ Handle immediate operand for SRS instruction
```

```
        1F CHECK# 0000001F OP-BUILD  RESET-# ;

PUBLIC

INSTRUCTIOND# srs   ( -- )   \ ARM Assembler, SRS instruction
        F84D0A00 , FE5FFFE0 , FFFFFFFF , ' SRS#  , 01A00000 ,

INSTRUCTIOND rfe   ( -- )   \ ARM Assembler, RFE instruction
        F8100A00 , FE50FFFF , FFFFFF10 , 01A00000 ,

PRIVATE

\ ------------------------------------------------------------------------

PUBLIC

INSTRUCTION ldrex   ( -- )   \ ARM Assembler, swap instruction
        E1900F9F , 0FF00FFF , FFFF100C ,

INSTRUCTION strex   ( -- )   \ ARM Assembler, swap instruction
        E1800F90 , 0FF00FF0 , FF10000C ,

PRIVATE

\ ------------------------------------------------------------------------

PUBLIC

INSTRUCTION umaal   ( -- )   \ ARM Assembler, multiply instruction
        E0400090 , 0FF000F0 , 0800100C ,

PRIVATE

\ ------------------------------------------------------------------------

: PAS-INSTRUCTION   ( x "<spaces>name" -- )   \ Parallel add/sub intruction
        CREATE ,
        DOES>   ( -- )
        @ 0FF00FF0 FF00100C OP-BEGIN
;

PUBLIC

E6100F10 PAS-INSTRUCTION sadd16   ( -- )   \ ARM Assembler, parallel add/sub
E6100F30 PAS-INSTRUCTION saddsubx   ( -- )   \ ARM Assembler, parallel add/sub
E6100F50 PAS-INSTRUCTION ssubaddx   ( -- )   \ ARM Assembler, parallel add/sub
E6100F70 PAS-INSTRUCTION ssub16   ( -- )   \ ARM Assembler, parallel add/sub
E6100F90 PAS-INSTRUCTION sadd8   ( -- )   \ ARM Assembler, parallel add/sub
E6100FF0 PAS-INSTRUCTION ssub8   ( -- )   \ ARM Assembler, parallel add/sub

E6200F10 PAS-INSTRUCTION qadd16   ( -- )   \ ARM Assembler, parallel add/sub
E6200F30 PAS-INSTRUCTION qaddsubx   ( -- )   \ ARM Assembler, parallel add/sub
E6200F50 PAS-INSTRUCTION qsubaddx   ( -- )   \ ARM Assembler, parallel add/sub
E6200F70 PAS-INSTRUCTION qsub16   ( -- )   \ ARM Assembler, parallel add/sub
E6200F90 PAS-INSTRUCTION qadd8   ( -- )   \ ARM Assembler, parallel add/sub
E6200FF0 PAS-INSTRUCTION qsub8   ( -- )   \ ARM Assembler, parallel add/sub

E6300F10 PAS-INSTRUCTION shadd16   ( -- )   \ ARM Assembler, parallel add/sub
E6300F30 PAS-INSTRUCTION shaddsubx   ( -- )   \ ARM Assembler, parallel add/sub
E6300F50 PAS-INSTRUCTION shsubaddx   ( -- )   \ ARM Assembler, parallel add/sub
E6300F70 PAS-INSTRUCTION shsub16   ( -- )   \ ARM Assembler, parallel add/sub
E6300F90 PAS-INSTRUCTION shadd8   ( -- )   \ ARM Assembler, parallel add/sub
E6300FF0 PAS-INSTRUCTION shsub8   ( -- )   \ ARM Assembler, parallel add/sub

E6500F10 PAS-INSTRUCTION uadd16   ( -- )   \ ARM Assembler, parallel add/sub
E6500F30 PAS-INSTRUCTION uaddsubx   ( -- )   \ ARM Assembler, parallel add/sub
E6500F50 PAS-INSTRUCTION usubaddx   ( -- )   \ ARM Assembler, parallel add/sub
E6500F70 PAS-INSTRUCTION usub16   ( -- )   \ ARM Assembler, parallel add/sub
E6500F90 PAS-INSTRUCTION uadd8   ( -- )   \ ARM Assembler, parallel add/sub
E6500FF0 PAS-INSTRUCTION usub8   ( -- )   \ ARM Assembler, parallel add/sub
```

```
E6600F10 PAS-INSTRUCTION uqadd16   ( -- )   \ ARM Assembler, parallel add/sub
E6600F30 PAS-INSTRUCTION uqaddsubx   ( -- )   \ ARM Assembler, parallel add/sub
E6600F50 PAS-INSTRUCTION uqsubaddx   ( -- )   \ ARM Assembler, parallel add/sub
E6600F70 PAS-INSTRUCTION uqsub16   ( -- )   \ ARM Assembler, parallel add/sub
E6600F90 PAS-INSTRUCTION uqadd8   ( -- )   \ ARM Assembler, parallel add/sub
E6600FF0 PAS-INSTRUCTION uqsub8   ( -- )   \ ARM Assembler, parallel add/sub

E6700F10 PAS-INSTRUCTION uhadd16   ( -- )   \ ARM Assembler, parallel add/sub
E6700F30 PAS-INSTRUCTION uhaddsubx   ( -- )   \ ARM Assembler, parallel add/sub
E6700F50 PAS-INSTRUCTION uhsubaddx   ( -- )   \ ARM Assembler, parallel add/sub
E6700F70 PAS-INSTRUCTION uhsub16   ( -- )   \ ARM Assembler, parallel add/sub
E6700F90 PAS-INSTRUCTION uhadd8   ( -- )   \ ARM Assembler, parallel add/sub
E6700FF0 PAS-INSTRUCTION uhsub8   ( -- )   \ ARM Assembler, parallel add/sub

PRIVATE

\ ------------------------------------------------------------------------

: PKH/SAT-LSR#   ( x -- )   \ Handle immediate operand for PKT & SAT shifts
        1F CHECK#  7 LSHIFT 00000F80 OP-BUILD  RESET-# ;

PUBLIC

: lsl   ( -- )   \ Extend LSL shift to cope with PKH & SAT instructions
        OP-VALUE @ 0F800070 AND 06800010 <>
        IF lsl EXIT THEN   \ if not PKH/SAT instruction do normal LSL
        00000000 00000040 OP-BUILD
        ['] PKH/SAT-LSR# '# !
;

PRIVATE

: PKH/SAT-ASR#   ( x -- )
        1- 1F CHECK# 1+  1F AND PKH/SAT-LSR# ;

PUBLIC

: asr   ( -- )   \ Extend ASR shift to cope with PKH & SAT instructions
        OP-VALUE @ 0F800030 AND 06800010 <>
        IF asr EXIT THEN   \ if not PKH/SAT instruction do normal ASR
        00000040 DUP OP-BUILD
        ['] PKH/SAT-ASR# '# !
;

INSTRUCTIOND pkhbt   ( -- )   \ ARM Assembler, PKHBT instruction
        E6800010 , 0FF00030 , FF00100C , F0000FC0 ,

: pkhtb   ( -- )   \ ARM Assembler, PKHTB instruction
        pkhbt ;

PRIVATE

: SAT#   ( x -- )   \ Handle immediate operand for SAT instructions
        1F CHECK#  10 LSHIFT 001F0000 OP-BUILD  RESET-# ;

: SAT16#   ( x -- )   \ Handle immediate operand for SAT16 instructions
        F CHECK#  10 LSHIFT 000F0000 OP-BUILD  RESET-# ;

PUBLIC

INSTRUCTIOND# ssat   ( -- )   \ ARM Assembler, SSAT instruction
        E6A00010 , 0FE00030 , FFFF000C , ' SAT# , F0000FC0 ,

INSTRUCTIOND# usat   ( -- )   \ ARM Assembler, USAT instruction
        E6E00010 , 0FE00030 , FFFF000C , ' SAT# , F0000FC0 ,

INSTRUCTION# ssat16   ( -- )   \ ARM Assembler, SSAT16 instruction
        E6A00F30 , 0FF00FF0 , FFFF000C , ' SAT16# ,

INSTRUCTION# usat16   ( -- )   \ ARM Assembler, USAT16 instruction
        E6E00F30 , 0FF00FF0 , FFFF000C , ' SAT16# ,
```

```
INSTRUCTION sel   ( -- )   \ ARM Assembler, SEL instruction
        E6800FB0 , 0FF00FF0 , FF00100C ,

PRIVATE

\ ----------------------------------------------------------------------------

: EXT#   ( x -- x)   \ Handle immediate operand for Extend instructions
        DUP FFFFFFE7 AND IF INVALID# THEN
        PKH/SAT-LSR#
;

PUBLIC

: ror   ( -- )   \ Extend ROR shift to cope with Extend instructions
        OP-VALUE @ 0F8000F0 AND 06800070 <>
        IF ror EXIT THEN   \ if not extend instruction do normal ROR
        ['] EXT# '# !
;

INSTRUCTIOND sxtab16   ( -- )   \ ARM Assembler, extend instruction
        E6800070 , 0FF00070 , FF00100C , F0000F80 ,

INSTRUCTIOND sxtab   ( -- )   \ ARM Assembler, extend instruction
        E6A00070 , 0FF00070 , FF00100C , F0000F80 ,

INSTRUCTIOND sxtah   ( -- )   \ ARM Assembler, extend instruction
        E6B00070 , 0FF00070 , FF00100C , F0000F80 ,

INSTRUCTIOND uxtab16   ( -- )   \ ARM Assembler, extend instruction
        E6C00070 , 0FF00070 , FF00100C , F0000F80 ,

INSTRUCTIOND uxtab   ( -- )   \ ARM Assembler, extend instruction
        E6E00070 , 0FF00070 , FF00100C , F0000F80 ,

INSTRUCTIOND uxtah   ( -- )   \ ARM Assembler, extend instruction
        E6F00070 , 0FF00070 , FF00100C , F0000F80 ,

INSTRUCTIOND sxtb16   ( -- )   \ ARM Assembler, extend instruction
        E68F0070 , 0FFF0070 , FFFF000C , F0000F80 ,

INSTRUCTIOND sxtb   ( -- )   \ ARM Assembler, extend instruction
        E6AF0070 , 0FFF0070 , FFFF000C , F0000F80 ,

INSTRUCTIOND sxth   ( -- )   \ ARM Assembler, extend instruction
        E6BF0070 , 0FFF0070 , FFFF000C , F0000F80 ,

INSTRUCTIOND uxtb16   ( -- )   \ ARM Assembler, extend instruction
        E6CF0070 , 0FFF0070 , FFFF000C , F0000F80 ,

INSTRUCTIOND uxtb   ( -- )   \ ARM Assembler, extend instruction
        E6EF0070 , 0FFF0070 , FFFF000C , F0000F80 ,

INSTRUCTIOND uxth   ( -- )   \ ARM Assembler, extend instruction
        E6FF0070 , 0FFF0070 , FFFF000C , F0000F80 ,

INSTRUCTION rev   ( -- )   \ ARM Assembler, reverse instruction
        E6BF0F30 , 0FFF0FF0 , FFFF000C ,

INSTRUCTION rev16   ( -- )   \ ARM Assembler, reverse instruction
        E6BF0FB0 , 0FFF0FF0 , FFFF000C ,

INSTRUCTION revsh   ( -- )   \ ARM Assembler, reverse instruction
        E6FF0FB0 , 0FFF0FF0 , FFFF000C ,

PRIVATE

\ ----------------------------------------------------------------------------

PUBLIC
```

```
INSTRUCTION smuad   ( -- )   \ ARM Assembler, multiply instruction
        E700F010 , 0FF0F0F0 , FF080010 ,

INSTRUCTION smuadx   ( -- )   \ ARM Assembler, multiply instruction
        E700F030 , 0FF0F0F0 , FF080010 ,

INSTRUCTION smusd   ( -- )   \ ARM Assembler, multiply instruction
        E700F050 , 0FF0F0F0 , FF080010 ,

INSTRUCTION smusdx   ( -- )   \ ARM Assembler, multiply instruction
        E700F070 , 0FF0F0F0 , FF080010 ,

INSTRUCTION smmul   ( -- )   \ ARM Assembler, multiply instruction
        E750F010 , 0FF0F0F0 , FF080010 ,

INSTRUCTION smmulr   ( -- )   \ ARM Assembler, multiply instruction
        E750F030 , 0FF0F0F0 , FF080010 ,

INSTRUCTION smlad   ( -- )   \ ARM Assembler, multiply instruction
        E7000010 , 0FF000F0 , 0C080010 ,

INSTRUCTION smladx   ( -- )   \ ARM Assembler, multiply instruction
        E7000030 , 0FF000F0 , 0C080010 ,

INSTRUCTION smlsd   ( -- )   \ ARM Assembler, multiply instruction
        E7000050 , 0FF000F0 , 0C080010 ,

INSTRUCTION smlsdx   ( -- )   \ ARM Assembler, multiply instruction
        E7000070 , 0FF000F0 , 0C080010 ,

INSTRUCTION smlald   ( -- )   \ ARM Assembler, multiply instruction
        E7400010 , 0FF000F0 , 0800100C ,

INSTRUCTION smlaldx   ( -- )   \ ARM Assembler, multiply instruction
        E7400030 , 0FF000F0 , 0800100C ,

INSTRUCTION smlsld   ( -- )   \ ARM Assembler, multiply instruction
        E7400050 , 0FF000F0 , 0800100C ,

INSTRUCTION smlsldx   ( -- )   \ ARM Assembler, multiply instruction
        E7400070 , 0FF000F0 , 0800100C ,

INSTRUCTION smmla   ( -- )   \ ARM Assembler, multiply instruction
        E7500010 , 0FF000F0 , 0C080010 ,

INSTRUCTION smmlar   ( -- )   \ ARM Assembler, multiply instruction
        E7500030 , 0FF000F0 , 0C080010 ,

INSTRUCTION smmls   ( -- )   \ ARM Assembler, multiply instruction
        E75000D0 , 0FF000F0 , 0C080010 ,

INSTRUCTION smmlsr   ( -- )   \ ARM Assembler, multiply instruction
        E75000F0 , 0FF000F0 , 0C080010 ,

PRIVATE

\ ------------------------------------------------------------------------

PUBLIC

INSTRUCTION usad8   ( -- )   \ ARM Assembler, USAD instruction
        E780F010 , 0FF0F0F0 , FF080010 ,

INSTRUCTION usada8   ( -- )   \ ARM Assembler, USAD instruction
        E7800010 , 0FF000F0 , 0C080010 ,

PRIVATE

\ ------------------------------------------------------------------------
```

```
PREVIOUS PREVIOUS PREVIOUS DEFINITIONS

DECIMAL
```