VisUAL
A highly visual ARM emulator

List of Supported Instructions

VisUAL supports a small subset of ARM UAL instructions. These are primarily arithmetic, logical, load/store and branch instructions. A short summary of the instruction syntax is given below. For detailed information and examples, press `Ctrl+Space` when typing an instruction opcode in the code editor.

Summary	Opcode	Syntax
Move	MOV	`MOV{S}{cond} dest, op1 {, SHIFT_op #expression}`
Move Negated	MVN	`MVN{S}{cond} dest, op1 {, SHIFT_op #expression}`
Address Load	ADR	`ADR{S}{cond} dest, expression`
LDR Psuedo-Instruction	LDR	`LDR{S}{cond} dest, =expression`
Add	ADD	`ADD{S}{cond} dest, op1, op2 {, SHIFT_op #expression}`
Add with Carry	ADC	`ADC{S}{cond} dest, op1, op2 {, SHIFT_op #expression}`
Subtract	SUB	`SUB{S}{cond} dest, op1, op2 {, SHIFT_op #expression}`
Subtract with Carry	SBC	`SBC{S}{cond} dest, op1, op2 {, SHIFT_op #expression}`
Reverse Subtract	RSB	`RSB{S}{cond} dest, op1, op2 {, SHIFT_op #expression}`
Reverse Subtract with Carry	RSC	`RSC{S}{cond} dest, op1, op2 {, SHIFT_op #expression}`
Bitwise And	AND	`AND{S}{cond} dest, op1, op2 {, SHIFT_op #expression}`
Bitwise Exclusive Or	EOR	`EOR{S}{cond} dest, op1, op2 {, SHIFT_op #expression}`
Bitwise Clear	BIC	`BIC{S}{cond} dest, op1, op2 {, SHIFT_op #expression}`
Bitwise Or	ORR	`ORR{S}{cond} dest, op1, op2 {, SHIFT_op #expression}`
Logical Shift Left	LSL	`LSL{S}{cond} dest, op1, op2`
Logical Shift Right	LSR	`LSR{S}{cond} dest, op1, op2`
Arithmetic Shift Right	ASR	`ASR{S}{cond} dest, op1, op2`
Rotate Right	ROR	`ROR{S}{cond} dest, op1, op2`
Rotate Right and Extend	RRX	`RRX{S}{cond} op1, op2`
Compare	CMP	`CMP{cond} op1, op2 {, SHIFT_op #expression}`
Compare Negated	CMN	`CMN{cond} op1, op2 {, SHIFT_op #expression}`
Test Bit(s) Set	TST	`TST{cond} op1, op2 {, SHIFT_op #expression}`
Test Equals	TEQ	`TEQ{cond} op1, op2 {, SHIFT_op #expression}`
Load Register	LDR	`LDR{B}{cond} dest, [source {, OFFSET}]` Offset addressing `LDR{B}{cond} dest, [source, OFFSET]!` Pre-indexed addressing `LDR{B}{cond} dest, [source], OFFSET` Post-indexed addressing
Store Register	STR	`STR{B}{cond} source, [dest {, OFFSET}]` Offset addressing `STR{B}{cond} source, [dest, OFFSET]!` Pre-indexed addressing `STR{B}{cond} source, [dest], OFFSET` Post-indexed addressing
Load Multiple Registers	LDM[dir]	`LDM[dir]{cond} source, {list of registers}`
Store Multiple Registers	STM[dir]	`STM[dir]{cond} dest, {list of registers}`
Branch	B	`B{cond} target`
Branch with Link	BL	`BL{cond} target`
Declare Word(s) in Memory	DCD	`name DCD value_1, value_2, ... value_N`
Declare Constant	EQU	`name equ expression`
Declare Empty Word(s) in Memory	FILL	`{name} FILL N` `N must be a multiple of 4`

Summary	Opcode	Syntax
Stop Emulation	END	`END{cond}`

For a comprehensive guide on these instructions and to see examples, browse the ARM Infocenter website here.

Notes

- For all instructions that require `dest`, `op1`, & `op2`, `dest` and `op1` must be registers.
- `expression` is a numerical constant or expression that evaluates to a 32-bit number. The operators `+`, `-` and `*` are allowed. A constant is a decimal number as a series of digits `0-9`, a hexadecimal number prefixed with `0x` or `&` or a binary number prefixed with `0b`. **An additional restriction is that this number must be creatble by rotating an 8-bit number right by an even number of bits within a 32-bit word.**
- For MOV / MVN, `dest` must be a register.
- For CMP / CMN / TST / TEQ, `op1` must be a register.
- `{...}` indicates optional code.
- `{cond}` refers to the **condition code**. An official list of instruction codes can be seen here.
- `{S}` is the **set bit**. If this is present, the status bits will be set. The exact mechanism of these differs for each instruction. See the instruction summary page on ARM Infocenter website for details here.
- `{, SHIFT_op #expression}` means a shift operation can be performed on `op2` as part of the same instruction. The shifted version of `op2` is then used as the second operand for the main instruction. This feature is part of the **flexible second operand** feature of ARM UAL.
  - `SHIFT_op` can be any one of LSL, LSR, ASR, ROR, RRX.
  - With the exception of RRX, `#expression` can be a register from R0 through R15 or any numerical expression.
  - `op2` cannot be R15 or PC if a shift is being used.
  - For MOV / MVN / ADR, `op1` is used instead.
- `{, OFFSET}` refers to the offset applied to the source address or destination address for load and store instructions respectively. It can be a register, a numerical expression, or a shifted register (like the flexible second operand discussed earlier).
- Load and Store Instructions indexing modes:
  - If `]` is present after the first operand, **post-indexing** is used. This means that the destination pointer is updated with the value (current value + offest) *after* the load/store operation is completed. You can observe this by example using the "Show Pointer" visualisation feature.
  - If `!` is present at the end, **pre-indexing** is used. This means the destination pointer is updated with the value (current value + offset) *before* the load/store operation is performed. You can observe this by example using the "Show Pointer" visualisation feature.
  - There are a total of 9 possible indexing modes. A good summary is provided here.
- `{B}` refers to **byte mode**. By default, the LDR / STR instructions load a word (32 bits) from the memory at the given address. If `B` is used, the byte at the given address is loaded instead. Note that the convention for byte addressing is **Little Endian**. You can observe memory access operations and compare word/byte modes using the "Show Memory" visualisation feature.
- For `LDM` and `STM` instructions, `[dir]` indicates stack direction. This can be one of full ascending `FA`, full descending `FD`, empty ascending `EA`, and empty descending `ED`. You can also use aliases for these as described in the link that follows. The operation of these instructions is provided in detail on this ARM Infocenter page.
- The `target` for a branch instruction `B` must be a label on a line. This instruction will cause the program to "jump", i.e. branch, to this line of code.
- Branch with link `BL` is identical to branch, with the additional function that the link register is set to point to the next line of code before the branch is performed. This can be used to return from a subroutine.
- For `EQU`, `expression` can be any numerical expression.

Disclaimer

The information provided on this page is given on an as is basis, for the purpose of supporting the VisUAL application and its users to debug code. It may be not be up to date with official ARM UAL guidelines at the time you are using it. Always up-to-date information can be found here.