

Where do the values that are fed into an operand come from when an instruction is executed using the three address style?

1

Why are registers faster than memory?

2

What is the one-address style of instruction?

3

Describe the load-store style of instruction.

4

How is code written in ARM Assembly run?

5

What is the instruction to load a value at a memory address into a register?

6

What is the instruction to store a value in a register to a memory address?

7

What is the instruction to sum two numbers?

8

- Implemented using a flip flop or some other very fast volatile storage (rather than smaller, cheaper SRAM).
- Situated inside the processor, so there's less distance for the data to travel, which takes less time.
- Fewer of them so address decoding takes less time
- Data doesn't need to be transferred over a bus.

Memory.

The resulting value is also copied to a destination address in memory.

2

1

The only operations on memory are load and store operations. This means each instruction is very fast and very simple, but there are many instructions.

Where only one memory address may be used in any one instruction. The other operands must be registers.

4

3

LDR <register> <memory_address_alias>

It is first assembled using an assembler into machine code. Then it is loaded into memory and executed sequentially.

6

5

ADD <destination_register> <operand_register1>
 <operand_register2>

STR <register> <memory_address_alias>

8

7

What is the instruction to branch upon a condition?

9

What does the program counter do? What register is it?

10

*When is the DEFW command executed. What does it do?
What's its syntax?*

11

What does DEFB do?

12

What's the syntax of DEFS?



13

What does STRB do? What is its syntax?

14

What does LDRB do? What is its syntax?

15

When using Little Endian, bytes are read from  to .

16

It is used to store the memory address of the next instruction to be executed.

It's register 15.

B <condition> <branch_name>

10

9

It is executed before the program runs.

DEFB stores a single byte in memory. If you give it a string, the whole string will be stored (in multiple bytes).

It stores a value at a memory address and assigns the address an alias.

<alias> DEFW <value_1>, ... <value_n>

12

11

Stores the lowest eight bits of a register into memory.

STRB <register> <memory_alias>

<alias> DEFS <number_of_bytes>, <value_of_bytes>

14

13

When using Little Endian, bytes are read from left to right.

Loads the lowest eight bits of a specific memory address into a register. The other bits in the register are set to zero.

LDRB <register> <memory_alias>

16

15

When using Big Endian, bytes are read from to .

17

How many bits are assigned to a literal in an ARM instruction

18

What command should be used to load a literal into a register

19

How do you load an address into a register so you can use it as a pointer?

20

What does `DEFW <number>` do?

21

What does `DEFB <value>` do?

22

What does `DEFS <size>, <fill>` do?

23

What does `ALIGN` do?

24

LDR <register> =<literal>

ADR <register>, <alias> Now the <register> will hold the memory location (and is therefore a pointer to) the alias.

Note, this is a pseudo instruction, that is converted to either MOV <register> #<literal> or it will define a constant and load that in from memory.

It reserves a byte(s) of memory with the value <value>. If a string is passed as the value, multiple bytes will be reserved, each with the value of a character.

It reserves a word of memory initialised to <number>

Leaves blank bytes in memory so that the next DEFINE command will start on a word boundary.

It reserves a block of memory <size> bytes long initialised to the value <fill>.

What does ENTRY do?

25

What does EQU do?

26

What are the four status flags provided by the ARM architecture?

27

How can you combine a CMP instruction with another instruction?

28

What does RSB do?

29

What does MLA do?

30

<i>Condition code</i>	<i>Meaning (for cmp or subs)</i>
-----------------------	----------------------------------

eq

ne

ge

le

lt

gt

31

In the following instruction, what method of addressing is used, what will R0 contain and what will happen to R1?

LDR R0, [R1]

32

EQU allows us to name a literal. You can then refer to the literal (still with a hash before it) by name in your code which is easy to read.

```
<alias> EQU #<value>
```

Sets the PC at the start of the program (i.e. dictates where the program should) start from.

26

25

Append S to an instruction. E.g.:

```
SUBS R0, R1, R2
```

If the result in R0 was negative, then the negative flag would be set etc etc.

- Negative
- Zero
- Carry
- Overflow

28

27

Multiply and add. MLA R0, R1, R2, R3:

```
R0 = (R0 * R1) + R2
```

Reverse subtract. RSB R1, R0, #0:

```
R1 = 0 - R0 = -R0
```

30

29

This is called **register-indirect** addressing.

The value loaded into R0 will be the 32 bits stored at the memory address that is equal to the value in R1.

R1 won't be altered at all.

Condition code	Meaning (for cmp or subs)
eq	Equal
ne	Not equal
ge	Signed greater than or equal
le	Signed less than or equal
lt	Signed less than
gt	Signed greater than

32

31

<p><i>In the following instruction, what method of addressing is used, what will R0 contain and what will happen to R1?</i></p> <p style="text-align: center;">LDR R0, [R1, #4]</p> <p style="text-align: right;">33</p>	<p><i>In the following instruction, what method of addressing is used, what will R0 contain and what will happen to R1?</i></p> <p style="text-align: center;">LDR R0, [R1, #4] !</p> <p style="text-align: right;">34</p>
<p><i>In the following instruction, what method of addressing is used, what will R0 contain and what will happen to R1?</i></p> <p style="text-align: center;">LDR R0, [R1], #4</p> <p style="text-align: right;">35</p>	<p><i>In the following instruction, what method of addressing is used, what will R0 contain and what will happen to R1 and R2?</i></p> <p style="text-align: center;">LDR R0, [R1, R2]</p> <p style="text-align: right;">36</p>
<p><i>In the following instruction, what method of addressing is used, what will R0 contain and what will happen to R1 and R2?</i></p> <p style="text-align: center;">LDR R0, [R1, R2, LSL, #2]</p> <p style="text-align: right;">37</p>	<p><i>How do you load a String into a register?</i></p> <p style="text-align: right;">38</p>
<p><i>What does the ADRL instruction do?</i></p> <p style="text-align: right;">39</p>	<p><i>How do you find the length of a string defined by the alias message?</i></p> <p style="text-align: right;">40</p>

*This is called **pre-indexed autoindexed addressing**.*

The value loaded into R0 will be the 32 bits stored at the memory address that is equal to the value in R1 + 4.

R1 will be incremented by 4 before the load operation.

34

*This is called **pre-indexed addressing**.*

The value loaded into R0 will be the 32 bits stored at the memory address that is equal to the value in R1 + 4.

R1 won't be altered at all.

33

*This is called **register-indexed addressing**.*

The value loaded into R0 will be the 32 bits stored at the memory address that is equal to the value in R1 + R2.

R1 and R2 will stay the same.

36

*This is called **post-indexed autoindexed addressing**.*

The value loaded into R0 will be the 32 bits stored at the memory address that is equal to the value in R1 + 4.

R1 will be incremented by 4 after the load operation.

35

```
msg DEFB "Hello"
ALIGN
ADRL R0, msg
SVC 3
```

*This is called **scaled register-indexed addressing**.*

*The value loaded into R0 will be the 32 bits stored at the memory address that is equal to the value in R1 + (R2 * 4).*

R1 and R2 will stay the same.

38

```
ADRL R1, message
MOV R2, #0
count LDRB R0, [R1, R2]
CMP R0, #0
ADDNE R2, R2, #1
BNE count
STR R2, length
```

It is a pseudo instruction that loads a program relative address into the register. It is compiled into two ADD instructions.

37

40

39

What are the four ARM bit shifting/rotation instructions?

41

What is the syntax of a shifting or rotation operation in ARM?

42

INSTRUCTION destination operand (#)shift

Mnemonic	Meaning
LSL	<i>Logical shift left</i>
LSR	<i>Logical shift right</i>
ASR	<i>Arithmetic shift right</i>
ROR	<i>Rotate Right</i>