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

Memory.

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

1

- *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.*

2

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

3

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.

4

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

5

`LDR <register> <memory_address_alias>`

6

`STR <register> <memory_address_alias>`

7

`ADD <destination_register> <operand_register1> <operand_register2>`

8

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 it's syntax?*

14

What does `LDRB` *do? What is it's syntax?*

15

When using Little Endian, bytes are read from [redacted] *to* [redacted].

16

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

It's register 15.

10

`B <condition> <branch_name>`

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 [redacted] *to* [redacted].

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

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

```
LDR R0, [R1]
```

20

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

```
LDR R0, [R1, #4]
```

21

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

```
LDR R0, [R1, #4]!
```

22

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

```
LDR R0, [R1], #4
```

23

In the following instruction, what method of indexing is used, what will `R0` *contain and what will happen to* `R1` *and* `R2`*?*

```
LDR R0, [R1, R2]
```

24

12

18

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

17

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.*

20

`LDR <register> =<literal>`

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.*

19

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.*

22

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.*

21

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.*

24

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.*

23

In the following instruction, what method of indexing is used, what will `R0` *contain and what will happen to* `R1` *and* `R2`*?*

```
LDR R0, [R1, R2, LSL, #2]
```

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.*