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.*

2

Memory.

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

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.

4

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

3

`LDR`

register

$memory_{a}ddress_{a}lias$

6

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

5

`ADD`

$destination_{r}egister$

$operand_{r}egister1$

$operand_{r}egister2$

8

`STR`

register

$memory_{a}ddress_{a}lias$

7

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

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

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

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

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

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

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

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

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]
```

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 **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.*

10

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.*

9

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.*

12

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.*

11

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.*

14

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.*

13