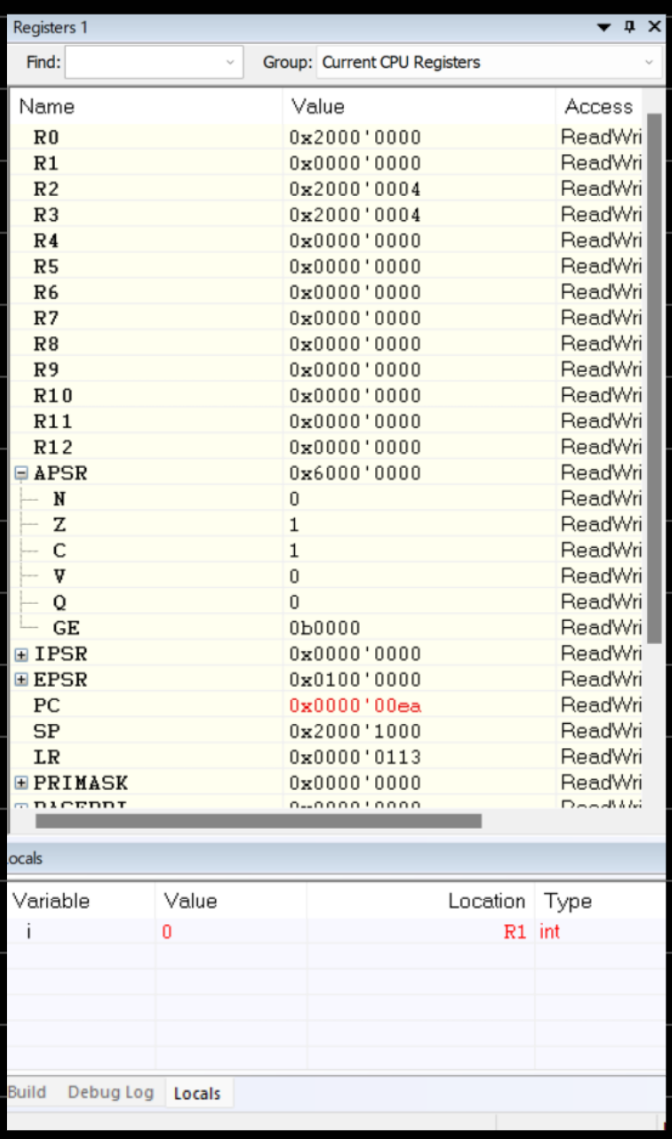
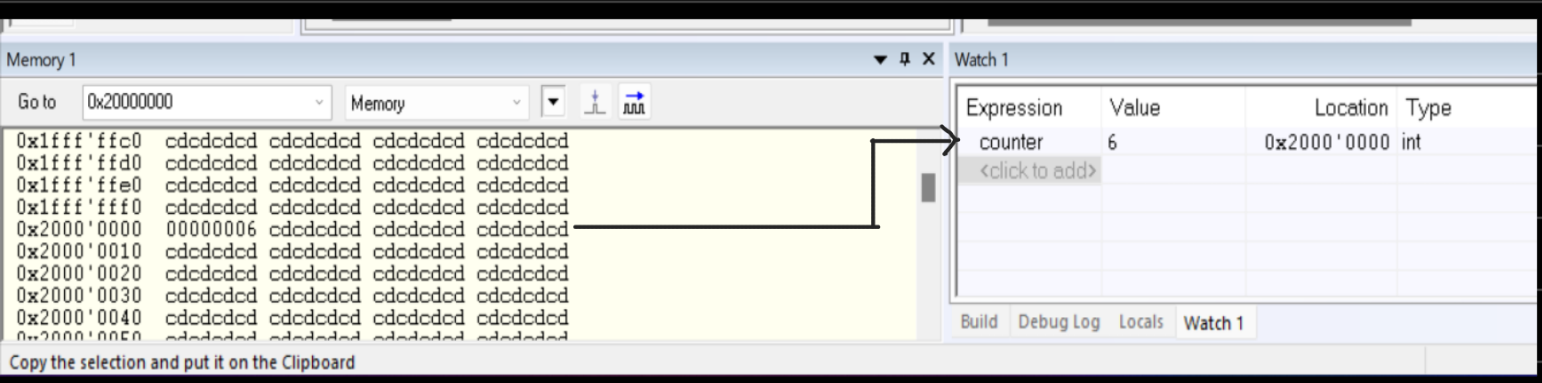
Global variables

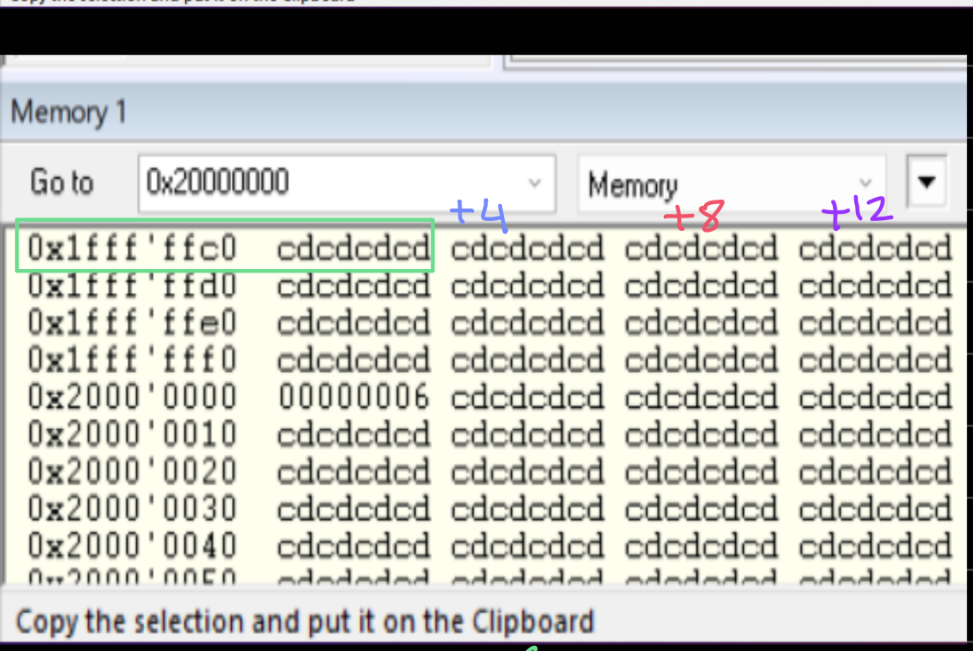
```
int counter = 0;

int main()
{
 counter = 1;

 for(int i = 0; i < 10; i++)
 {
  counter++;
 }

 return 0;
}
```

→ counter has been placed outside as a global variable. As such, it does not appear in the local variables list.

To find the global variable, we switch to "watch 1" from "view".
We search for counter there.
Counter is now saved in memory address 0x20000000 (can be random) and increments from there.







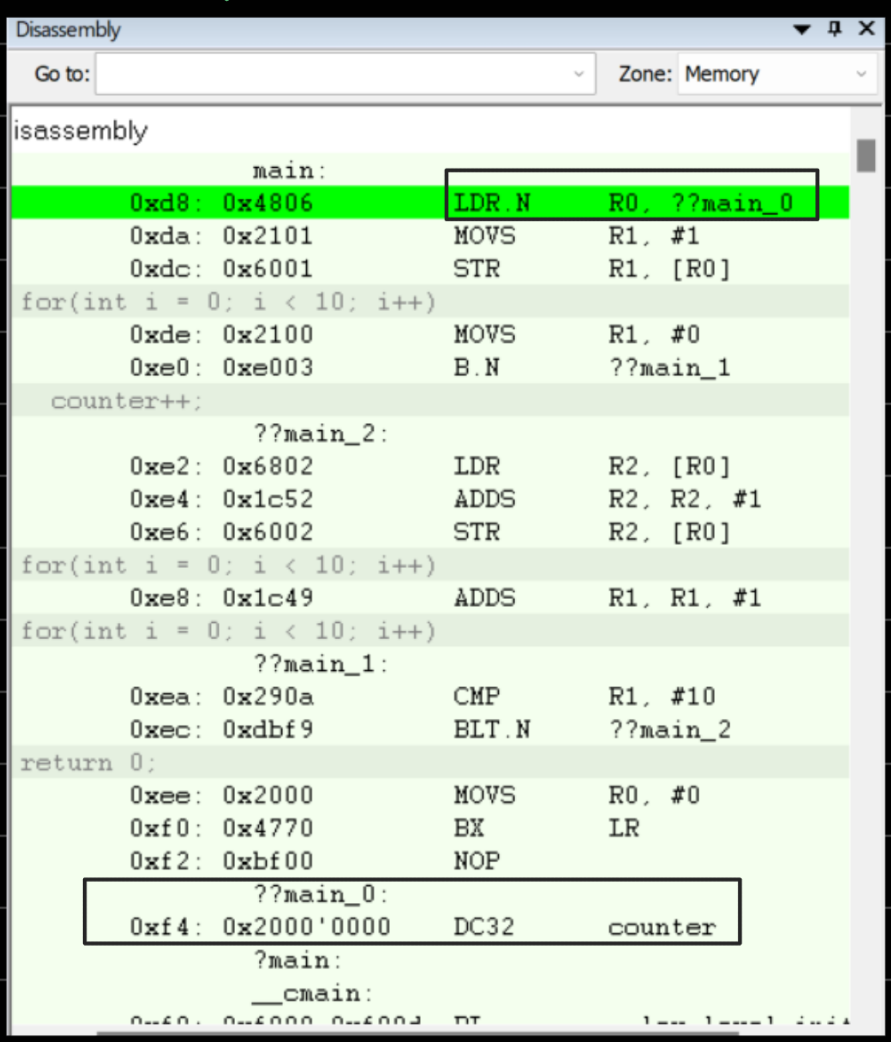
every row is an offset of 0x00000010 of previous row

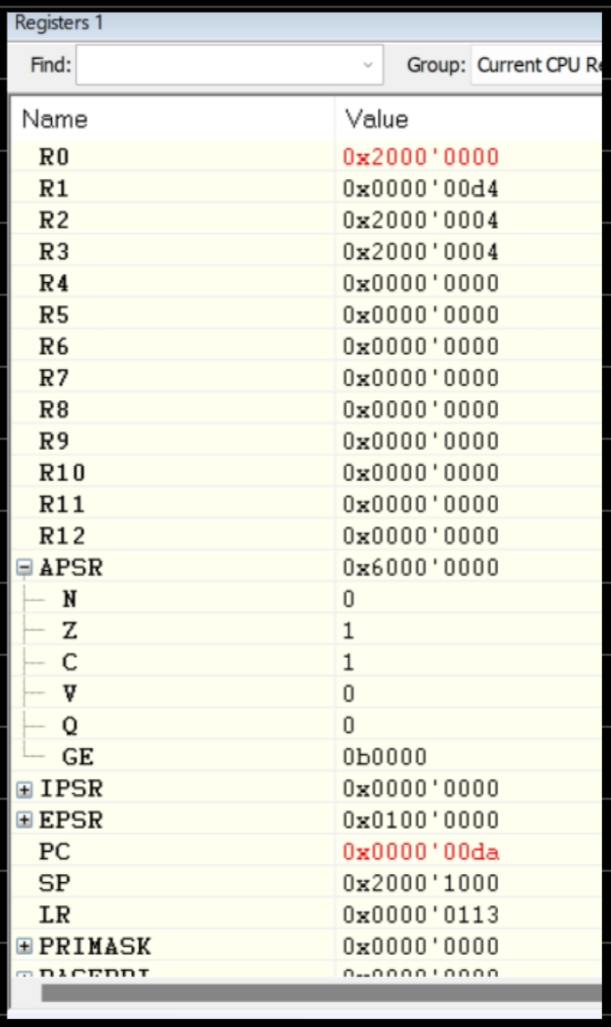
Address is in left most column. 2nd column shows the data in the address of the left column, 3rd column shows data in the rows address + 0x00000004 offset.

4th column shows data in the rows address + 0x00000008 offset

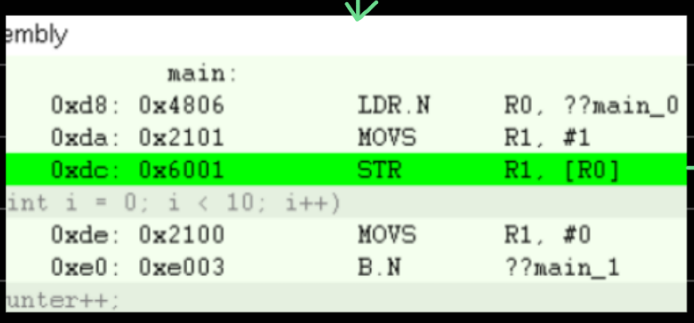
5th column shows data in the rows address + 0x0000000C offset

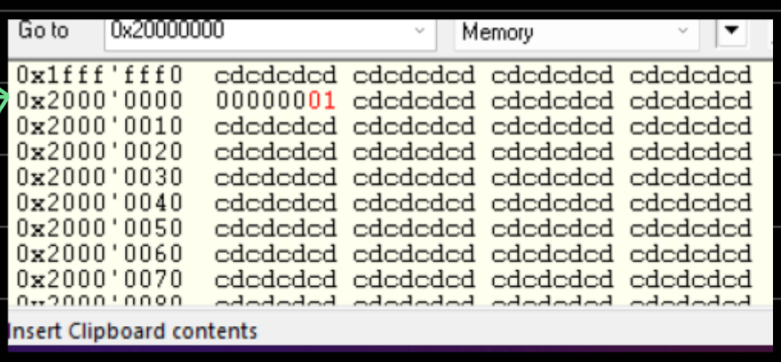
When main starts in the disassembly window, we can see the instruction LDR.N (load from memory to a register) load the register R0 from the label "??main_0". This label found in the disassembly window contains the address 0x20000000 (which is address of counter) Thus R0 is loaded with 0x20000000





Next register R1 is loaded with number 1. The next STR R1,[R0] instruction saves the data in R1 to the memory pointed at by R0 (location 0x20000000)

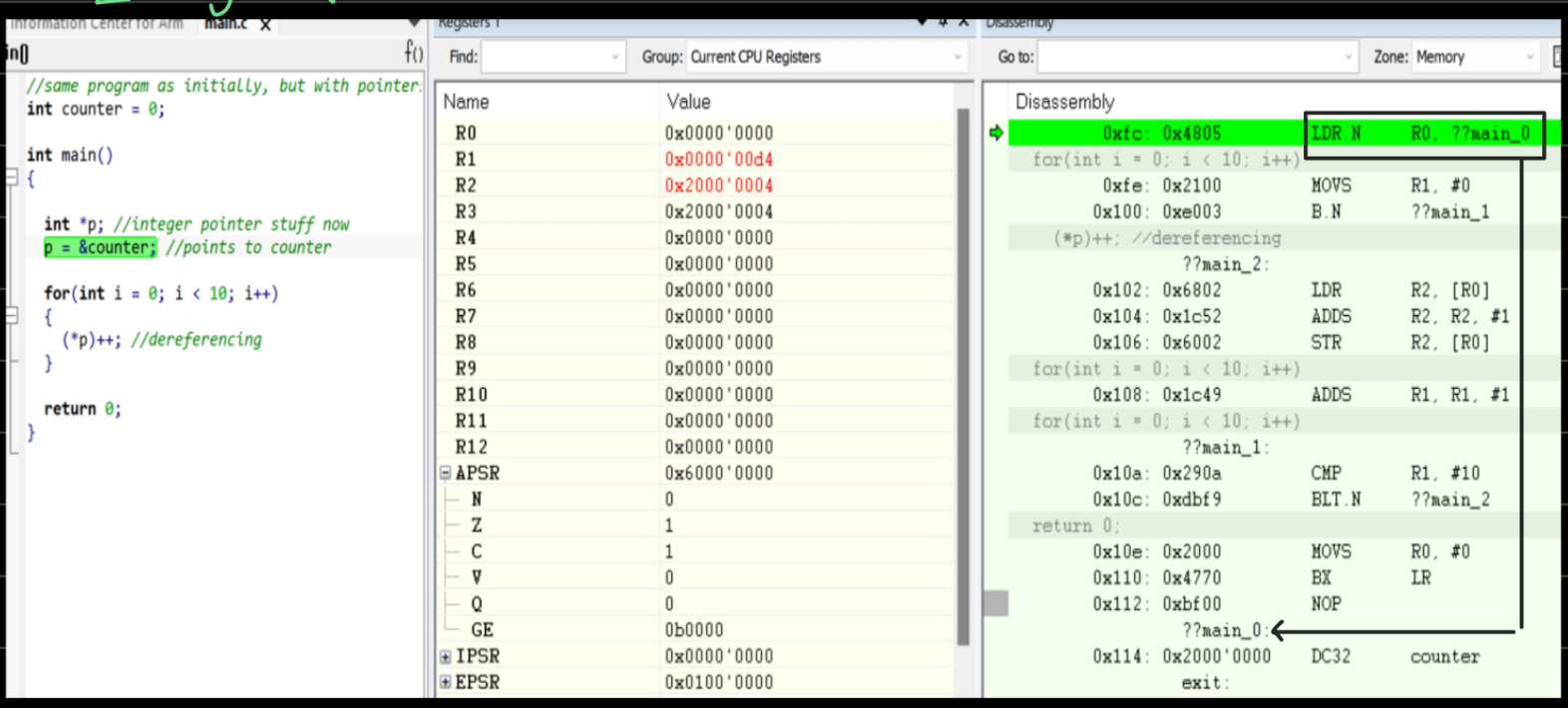




This is RISC architecture where all data modification must happen in registers. STR then writes to memory

```
//same program as initially, but with pointers
int counter = 0;

int main()
{

 int *p; //integer pointer stuff now
 p = &counter; //points to counter

 for(int i = 0; i < 10; i++)
 {
  (*p)++; //dereferencing
 }

 return 0;
}
```

With this program we create a local pointer p that holds address 0x2000 0000 (counter). P is stored in R0 register and is held constant. Because R0 is not used, we now start seeing data manipulations in R2 register.



Pointers increase machine efficiency.