

Global variables

```
int counter = 0;           → counter has been placed outside as a
int main()                global variable. As such, it does not appear
{                         in the local variables list.
    counter = 1;
    for(int i = 0; i < 10; i++)
    {
        counter++;
    }
    return 0;
}
```

To find the global variable, we switch to "watch 1" from "view".

We search for counter there. Counter is now saved in memory address 0x2000'0000 (can be random) and increments from there.

Registers 1		
Name	Value	Access
R0	0x2000'0000	ReadWri
R1	0x0000'0000	ReadWri
R2	0x2000'0004	ReadWri
R3	0x2000'0004	ReadWri
R4	0x0000'0000	ReadWri
R5	0x0000'0000	ReadWri
R6	0x0000'0000	ReadWri
R7	0x0000'0000	ReadWri
R8	0x0000'0000	ReadWri
R9	0x0000'0000	ReadWri
R10	0x0000'0000	ReadWri
R11	0x0000'0000	ReadWri
R12	0x0000'0000	ReadWri
APSR	0x6000'0000	ReadWri
N	0	ReadWri
Z	1	ReadWri
C	1	ReadWri
V	0	ReadWri
Q	0	ReadWri
GE	0b0000	ReadWri
IPSR	0x0000'0000	ReadWri
EPSR	0x0100'0000	ReadWri
PC	0x0000'00ea	ReadWri
SP	0x2000'1000	ReadWri
LR	0x0000'0113	ReadWri
PRIMASK	0x0000'0000	ReadWri
DIER	0x0000'0000	ReadWri
Locals		
Variable	Value	Location Type
i	0	R1 int

Memory 1			
Go to	Memory		
0xffff'fffc0	cdcdcdcd cdcdcdcd cdcdcdcd		
0xffff'ffd0	cdcdcdcd cdcdcdcd cdcdcdcd		
0xffff'ffe0	cdcdcdcd cdcdcdcd cdcdcdcd		
0xffff'fff0	cdcdcdcd cdcdcdcd cdcdcdcd		
0x2000'0000	00000006 cdcdcdcd cdcdcdcd		
0x2000'0010	cdcdcdcd cdcdcdcd cdcdcdcd		
0x2000'0020	cdcdcdcd cdcdcdcd cdcdcdcd		
0x2000'0030	cdcdcdcd cdcdcdcd cdcdcdcd		
0x2000'0040	cdcdcdcd cdcdcdcd cdcdcdcd		
0x2000'0050	adadadad adadadad adadadad		

Watch 1			
Expression	Value	Location	Type
counter	6	0x2000'0000	int

Memory 1			
Go to	Memory		
0xffff'fffc0	cdcdcdcd cdcdcdcd cdcdcdcd	+4	
0xffff'ffd0	cdcdcdcd cdcdcdcd cdcdcdcd	+8	
0xffff'ffe0	cdcdcdcd cdcdcdcd cdcdcdcd	+12	
0xffff'fff0	cdcdcdcd cdcdcdcd cdcdcdcd		
0x2000'0000	00000006 cdcdcdcd cdcdcdcd		
0x2000'0010	cdcdcdcd cdcdcdcd cdcdcdcd		
0x2000'0020	cdcdcdcd cdcdcdcd cdcdcdcd		
0x2000'0030	cdcdcdcd cdcdcdcd cdcdcdcd		
0x2000'0040	cdcdcdcd cdcdcdcd cdcdcdcd		
0x2000'0050	adadadad adadadad adadadad		

every row is an offset of 0x0000 0010 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 + 0x0000 00004 offset. 4th column shows data in the rows address + 0x0000 00008 offset. 5th column shows data in the rows address + 0x0000 0000c 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 0x2000000 (which is address of counter). Thus R0 is loaded with 0x2000 0000



Name	Value
R0	0x2000'0000
R1	0x0000'00d4
R2	0x2000'0004
R3	0x2000'0004
R4	0x0000'0000
R5	0x0000'0000
R6	0x0000'0000
R7	0x0000'0000
R8	0x0000'0000
R9	0x0000'0000
R10	0x0000'0000
R11	0x0000'0000
R12	0x0000'0000
APSR	0x6000'0000
N	0
Z	1
C	1
V	0
Q	0
GE	0b0000
IPSR	0x0000'0000
EPSR	0x0100'0000
PC	0x0000'00da
SP	0x2000'1000
LR	0x0000'0113
PRIMASK	0x0000'0000
SIDERRPT	0x0000'0000

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 0x2000 0000)



Goto	Memory
0x2000'0000	cdcdcdcd cdcdcdcd cdcdcdcd cdcdcdcd 0x2000'0000 00000001 cdcdcdcd cdcdcdcd cdcdcdcd 0x2000'0010 cdcdcdcd cdcdcdcd cdcdcdcd cdcdcdcd 0x2000'0020 cdcdcdcd cdcdcdcd cdcdcdcd cdcdcdcd 0x2000'0030 cdcdcdcd cdcdcdcd cdcdcdcd cdcdcdcd 0x2000'0040 cdcdcdcd cdcdcdcd cdcdcdcd cdcdcdcd 0x2000'0050 cdcdcdcd cdcdcdcd cdcdcdcd cdcdcdcd 0x2000'0060 cdcdcdcd cdcdcdcd cdcdcdcd cdcdcdcd 0x2000'0070 cdcdcdcd cdcdcdcd cdcdcdcd cdcdcdcd 0x2000'0080 cdcdcdcd cdcdcdcd cdcdcdcd cdcdcdcd

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.

The screenshot shows the Information Center for ARM tool interface. On the left, the source code is displayed:

```
//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;  
}
```

In the center, the Registers window shows the current CPU registers:

Name	Value
R0	0x0000'0000
R1	0x0000'00d4
R2	0x2000'0004
R3	0x2000'0004
R4	0x0000'0000
R5	0x0000'0000
R6	0x0000'0000
R7	0x0000'0000
R8	0x0000'0000
R9	0x0000'0000
R10	0x0000'0000
R11	0x0000'0000
R12	0x0000'0000
APSR	0x6000'0000
N	0
Z	1
C	1
V	0
Q	0
GE	0b0000
IPSR	0x0000'0000
EPSR	0x0100'0000

On the right, the Disassembly window shows the assembly code:

```
0xfc: 0x4805 LDR N R0, ??main_0  
for(int i = 0; i < 10; i++)  
    0xfe: 0x2100 MOVS R1, #0  
    0x100: 0xe003 B.N ??main_1  
    (*p)++; //dereferencing  
    ??main_2:  
    0x102: 0x6802 LDR R2, [R0]  
    0x104: 0x1c52 ADDS R2, R2, #1  
    0x106: 0x6002 STR R2, [R0]  
    for(int i = 0; i < 10; i++)  
        0x108: 0x1c49 ADDS R1, R1, #1  
        ??main_1:  
        0x10a: 0x290a CMP R1, #10  
        0x10c: 0xdbe9 BLT.N ??main_2  
    return 0;  
    0x10e: 0x2000 MOVS R0, #0  
    0x110: 0x4770 BX LR  
    0x112: 0xbff0 NOP  
    ??main_0: ←  
    0x114: 0x2000'0000 DC32 counter  
exit:
```

Pointers increase machine efficiency.