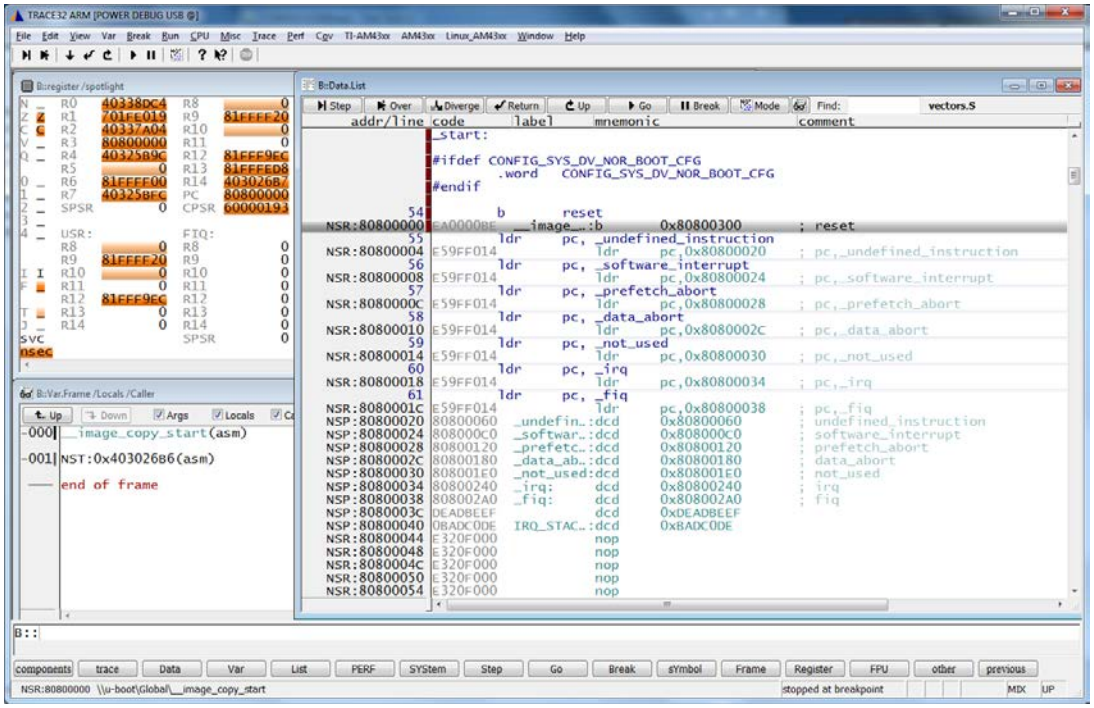
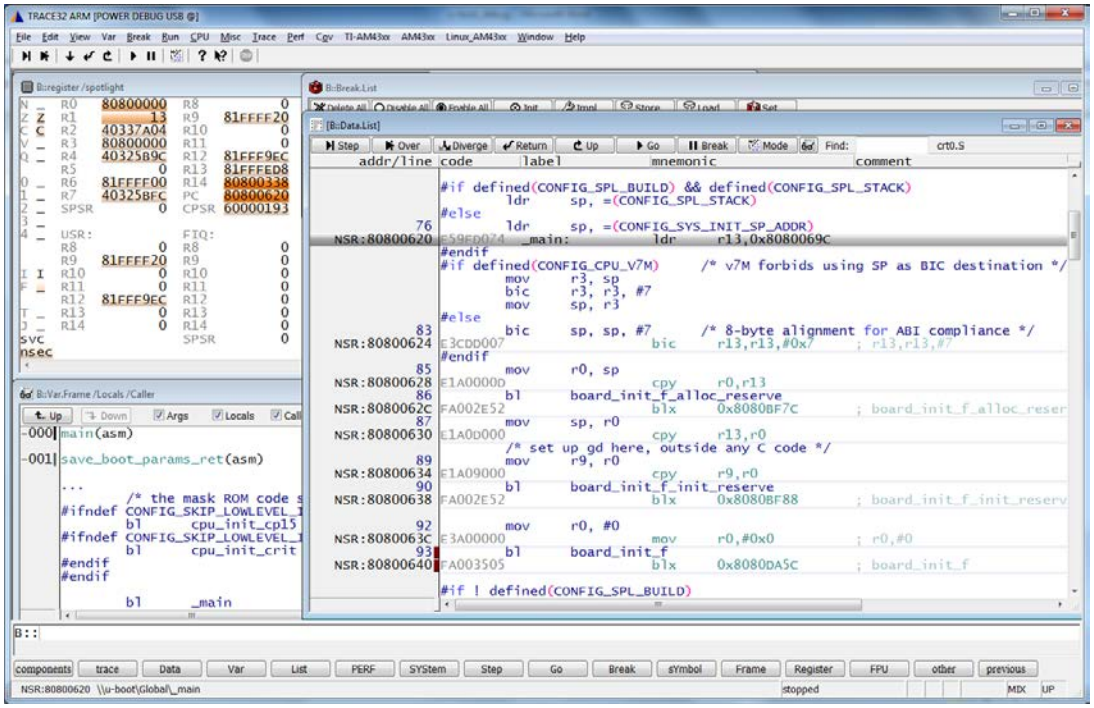
How to JTAG debug u-boot with relocation

1. At the u-boot entry.



2. At the entry of ENTRY(_main) in crt0.S for C-runtime startup Code for ARM U-Boot.

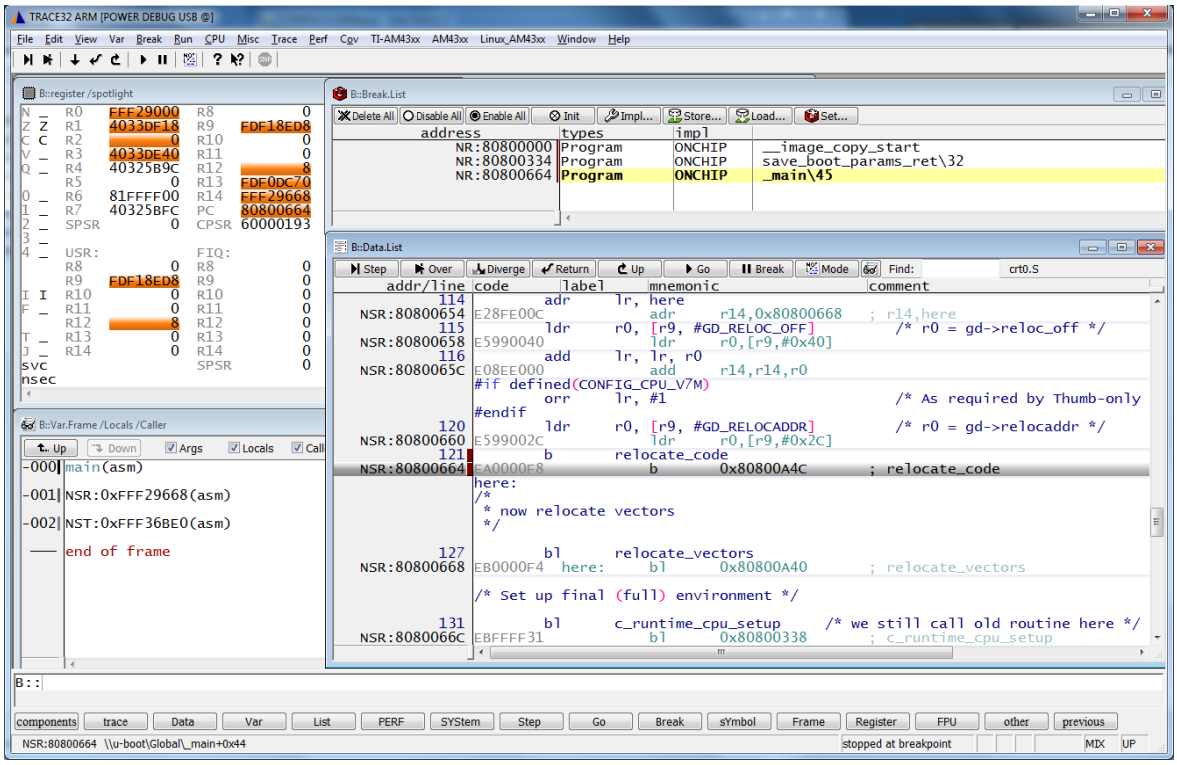


```
/*
 * This file handles the target-independent stages of the U-Boot
 * start-up where a C runtime environment is needed. Its entry point
 * is _main and is branched into from the target's start.S file.
 *
 * _main execution sequence is:
 *
 * 1. Set up initial environment for calling board_init_f().
 *    This environment only provides a stack and a place to store
 *    the GD ('global data') structure, both located in some readily
 *    available RAM (SRAM, locked cache...). In this context, VARIABLE
 *    global data, initialized or not (BSS), are UNAVAILABLE; only
 *    CONSTANT initialized data are available. GD should be zeroed
 *    before board_init_f() is called.
 *
 * 2. Call board_init_f(). This function prepares the hardware for
 *    execution from system RAM (DRAM, DDR...) As system RAM may not
 *    be available yet, , board_init_f() must use the current GD to
 *    store any data which must be passed on to later stages. These
 *    data include the relocation destination, the future stack, and
 *    the future GD location.
 *
 * 3. Set up intermediate environment where the stack and GD are the
 *    ones allocated by board_init_f() in system RAM, but BSS and
 *    initialized non-const data are still not available.
 *
 * 4a.For U-Boot proper (not SPL), call relocate_code(). This function
 *    relocates U-Boot from its current location into the relocation
 *    destination computed by board_init_f().
 *
 * 4b.For SPL, board_init_f() just returns (to crt0). There is no
 *    code relocation in SPL.
 *
 * 5. Set up final environment for calling board_init_r(). This
 *    environment has BSS (initialized to 0), initialized non-const
 *    data (initialized to their intended value), and stack in system
 *    RAM (for SPL moving the stack and GD into RAM is optional - see
 *    CONFIG_SPL_STACK_R). GD has retained values set by board_init_f().
 *
 * 6. For U-Boot proper (not SPL), some CPUs have some work left to do
 *    at this point regarding memory, so call c_runtime_cpu_setup.
 *
 * 7. Branch to board_init_r().
 *
 * For more information see 'Board Initialisation Flow in README.
 */
```

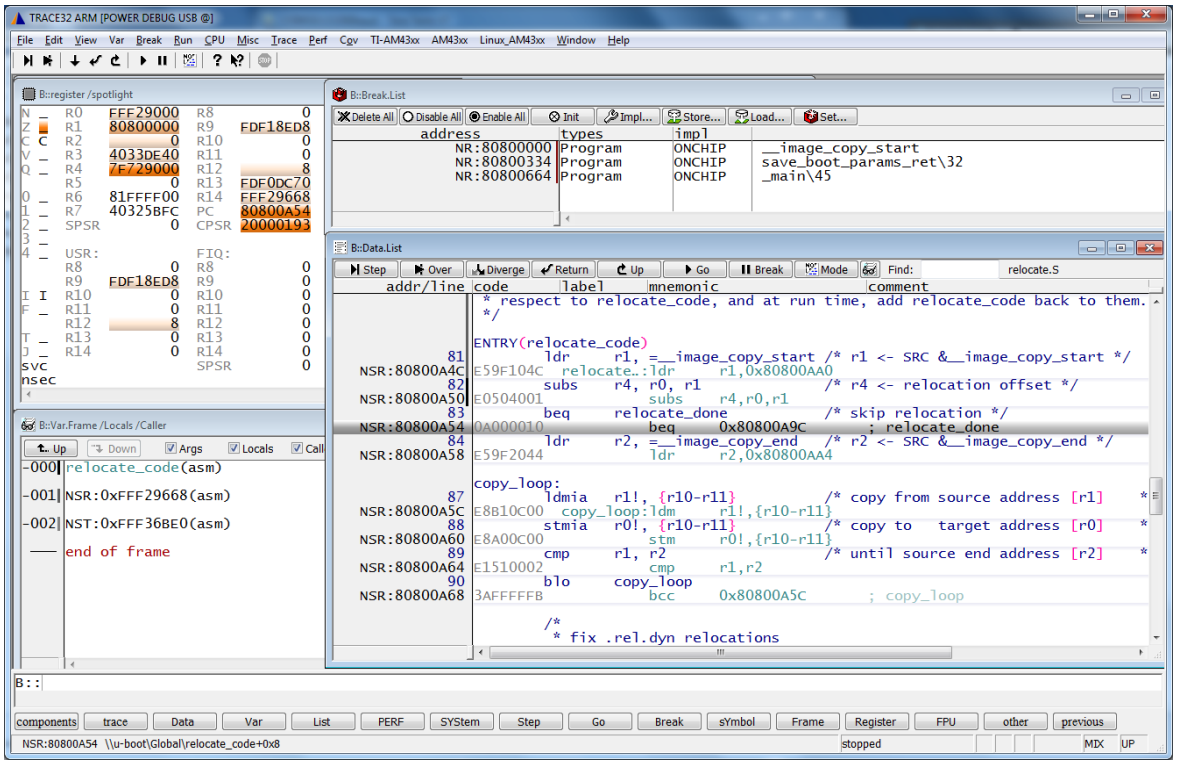
Note that the very first SPL serial output as shown below is from board_init_f().

```
U-Boot 2017.01-00319-g42b46bd-dirty (Aug 05 2020 - 12:17:33 -0700)
CPU  : AM437X-HS rev 1.2
Model: TI AM437x GP EVM
DRAM:  2 GiB
```

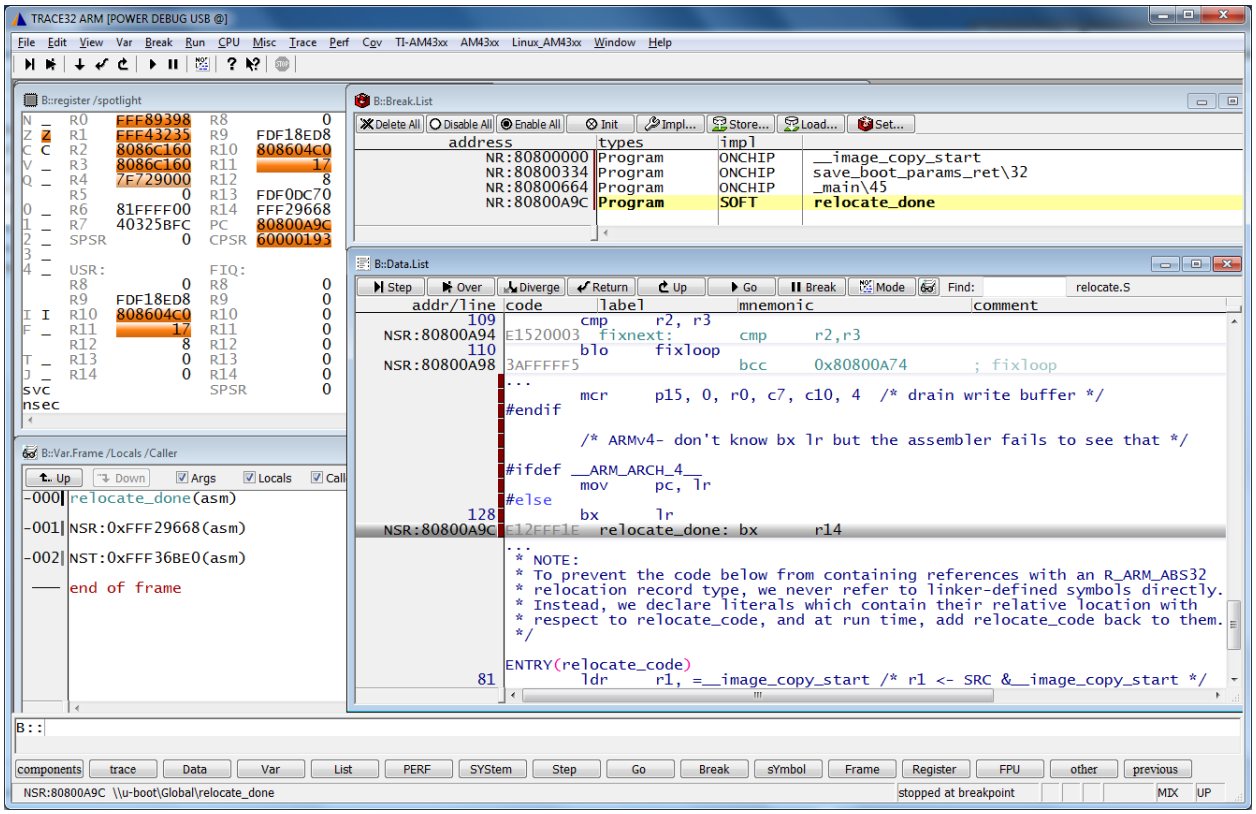
3. Set bkpt @relocate_code:



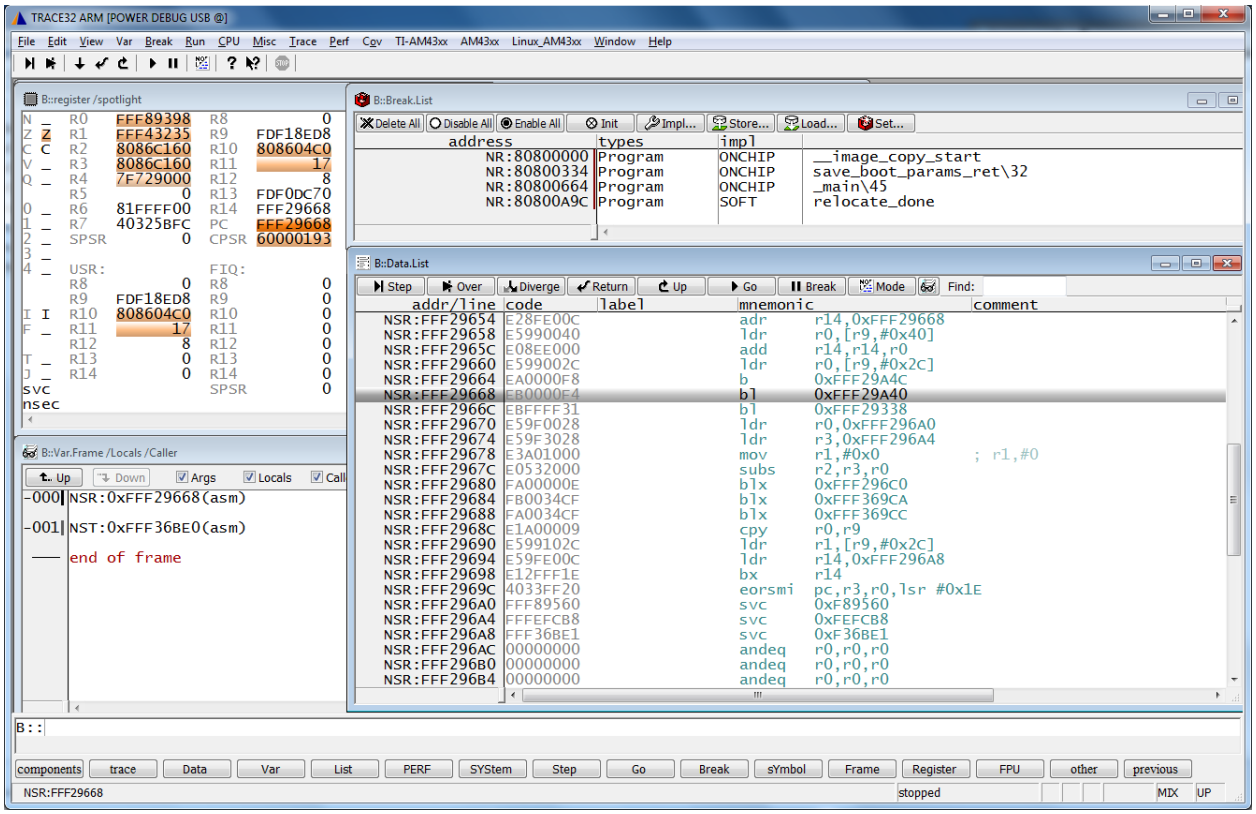
4. Step into relocate_code.
The relocation offset is in R4=0x7F729000 in this example



5. At the exit of relocate_code, this is the last instruction of u-boot running w/o relocation.



6. One step further, u-boot running after relocation



7. Relocate the u-boot symbol using <y.reloc 0x7F729000> for T32 JTAG debugger.
   The screenshot in #6 is shown below which matches u-boot SRC nicely.