MicroTESK Simulator Memory Configuration

Addresses used by MicroTESK for code and data section can be configured using the following settings:

- $base\_virtual\_address$ (by default, equals 0);
- $base\_physical\_address$ (by default, equals 0).

The settings are initialized in the $initialize$ method of a template in the following way:

```
def initialize
  super
  # Memory-related settings
  @base_virtual_address = 0x00001000
  @base_physical_address = 0x00001000
end
```

Address allocation for code and data sections is done using different mechanisms and must be considered separately.

Code Sections

NOTE: MicroTESK does not perform memory modeling for code sections. Instructions are not stored in physical memory and have no physical addresses. They use only virtual addresses.

MicroTESK starts address allocations for code sections at $base\_virtual\_address$. The VA for the current allocation is calculated as VA for the previous allocation + size of previous allocation.

Allocation address can be modified using the ***org*** directive. MicroTESK allows specifying **relative** and **absolute** origins:

- ***Relative origin***: `org :delta => n`, VA = VA + n
- ***Absolute origin***: `org n`, VA = $base\_virtual\_address$ + n.

Addresses can be aligned using the ***align*** directive. By default, `align n` means align at the border of 2^n bytes.

Data Sections

There two use cases of addressing data which are handled using separate mechanisms:

- Data is allocated in memory using the **`data`**`{...}` construct.
- Data is read or written to memory using load/store instructions.

Loads and stores use the MMU model included into the MicroTESK simulator, which involves address translation and accesses to cache buffers according to the MMU specifications.

Data sections do not use the MMU model. They place data directly to physical memory starting from $base\_physical\_address$. Address allocation is controlled by the ***org*** and ***align*** directives. They work similarly to the one used for code sections, but operate with physical addresses:

- ***Relative origin***: `org :delta => n`, PA = PA + n
- ***Absolute origin***: `org n`, PA = $base\_physical\_address$ + n.

Labels in data sections are assigned virtual addresses. To do this, PA is translated into VA. Address translation does not use the MMU model. Instead, it uses a simplified scheme based on settings that works as follows:

VA = $base\_virtual\_address$ + (PA - $base\_physical\_address$)

NOTE: Allocation addresses VA and PA are tracked separately for data and code sections. Therefore, it is required to take care to avoid address conflicts.

MicroTESK allows using different base addresses for code and data allocations. Data allocations can be assigned a separate base virtual address. This can be done in the following way (see the `pre` method of ArmV8BaseTemplate):

```
data_config(:text=>'.data', :target=>'M', :base_virtual_address=>0x00002000){
  ...
}
```

This VA is translated into PA using the simplified address translation scheme (PA = $base\_physical\_address$ + (VA - $base\_virtual\_address$)) and the result is used as base PA for data allocations.

Method get_address_of

The `get_address_of` method returns a virtual address associated with a label in a global data section. The correctness of this address is an issue of correctness of the above-described settings and address translation logic.

Example: min_max.rb

NOTE: This is a simplified example. It considers PA to be equal VA. This assumption is made because the address translation mechanism is currently disabled in MMU specifications for ARMv8. Anyway, this should be enough to illustrate the above-described principles.

Settings:

Setting values are as follows:

- $base\_virtual\_address$ and $base\_physical\_address$ use the default value 0 (are not initialized).
- $base\_virtual\_address$ for data sections is specified as `0x40180000` (see code below).

```
data_config(:text=>'.data', :target=>'M', :base_virtual_address=>0x40180000){
  ...
}
```

Prologue:

Staring VA for code section is specified as `0x2000` (*base_virtual_address* = 0 + origin = `0x2000`):

```
text '.text'
text '.globl _start'
movz x0, vbar_el1_value, 0
msr vbar_el1, x0
bl :_start
org 0x2000
label :_start
```

Data section:

Data sections starts at VA = `0x40180000` (*base_virtual_address* =`0x40180000 + origin=0x0`). VA of :***data*** is `0x40180000` and VA of :**end** is `0x40180060`:

```
data {
  org 0x0
  label :data
  dword rand(1, 0xffff), rand(1, 0xffff), rand(1, 0xffff), rand(1, 0xffff),
        rand(1, 0xffff), rand(1, 0xffff), rand(1, 0xffff), rand(1, 0xffff),
        rand(1, 0xffff)
  label :end
  space 1
}
```

Main code:

Code generated for the test case was assigned the following addresses:

```
0x0000000000002000 adr x0, data
0x0000000000002004 adr x1, end
0x0000000000002008 mov x2, x0
0x000000000000200c ldar x6, [x2, #0]
0x0000000000002010 mov x3, x6
0x0000000000002014 mov x4, x6
for_0000:
0x0000000000002018 cmp x2, x1, LSL #0
0x000000000000201c b.ge exit_for_0000
0x0000000000002020 ldar x6, [x2, #0]
0x0000000000002024 cmp x6, x4, LSL #0
0x0000000000002028 b.gt new_max_value_0000
0x000000000000202c cmp x6, x3, LSL #0
0x0000000000002030 b.lt new_min_value_0000
next_0000:
0x0000000000002034 add x2, x2, #8, LSL #0
0x0000000000002038 b for_0000
new_max_value_0000:
0x000000000000203c mov x4, x6
0x0000000000002040 b next_0000
new_min_value_0000:
0x0000000000002044 mov x3, x6
0x0000000000002048 b next_0000
exit_for_0000:
```