Debugger on micro-architecture Model

Titto Thomas

May 20, 2016

1 Introduction

Debugger on the micro-architecture model is an extension of the system designed for ISA-C model. The GDB-server bridge and it's interface with the hardware has been kept same, just the hardware implementation is changed.

The micro-architecture model described in Aa language is the accurate representation of final hardware. It has four main parts as shown in the figure : CPU Control Unit (CCU), Thread Execution Unit (TEU) and Memory Interface Unit (MIU). CCU generates the control signals for the other blocks and takes care of exception handling, whereas MIU facilitates all the memory accesses. TEU is the actual processor pipeline that will infinitely fetch and execute instruction until any exceptions occur. The Debug daemon facilitates debugging on AJIT processor with a GDB front end.

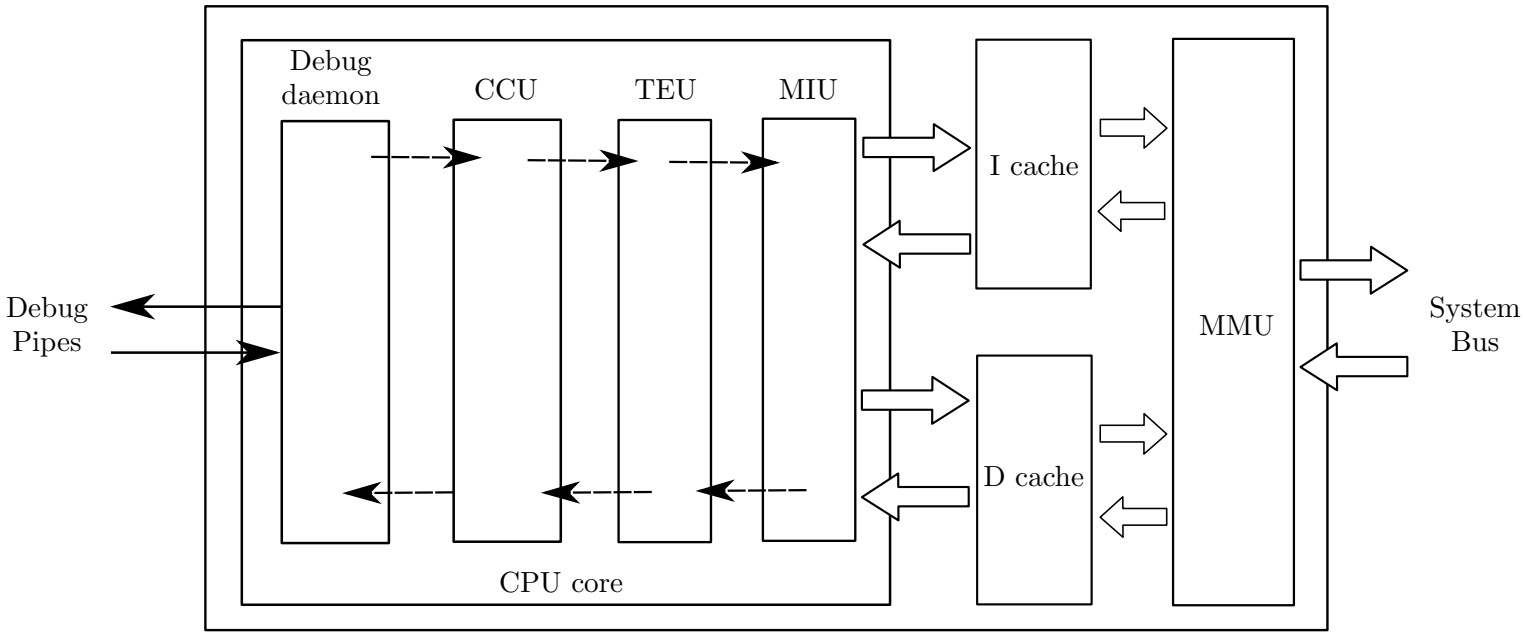

Figure 1: Micro-architecture model

2 Expected behaviours

Each of these 3 important blocks need to work in an expected manner for their correct functioning. These expected behaviours are represented in the form of flowcharts in the following sections.

2.1 Pseudo code : Debug daemon

Algorithm 1 Debug daemon

```
1: function DEBUG_DAEMON
2:     while ccu_msg ≠ CONNECT_RQST do                ▷ Check connect request from CCU
3:         ccu_msg ← ccu_to_debug_pipe
4:     end while
5:     while gdb_msg ≠ CONNECT_RQST do                ▷ Check connect request from GDB
6:         gdb_msg ← ENV_to_AJIT_pipe
7:     end while
8:     debug_to_ccu_pipe ← Acknowledge_OK
9:     AJIT_to_ENV_pipe ← Acknowledge_OK
10:    ccu_msg ← ccu_to_debug_pipe
11:    if (ccu_msg is valid) then
12:        Decode ccu_msg
13:        if (breakpoint or watchpoint or trap hit) then
14:            stored_PC ← ccu_to_debug_pipe
15:            stored_NPC ← ccu_to_debug_pipe
16:            stored_PSR ← ccu_to_debug_pipe
17:        end if
18:        ccu_to_gdb_msg ← Encode the message for GDB
19:        AJIT_to_ENV_pipe ← ccu_to_gdb_msg
20:        if (watchpoint(x) hit) then
21:            ccu_to_gdb_address ← stored adddress of x
22:            AJIT_to_ENV_pipe ← ccu_to_gdb_address
23:        end if
24:    end if
25:    gdb_msg ← ENV_to_AJIT_pipe
26:    if (gdb_msg is valid) then
27:        Decode gdb_msg
28:        debug_to_ccu_pipe ← gdb_msg
29:        if (msg_length = 2) then
30:            gdb_msg_2 ← ENV_to_AJIT_pipe
31:            debug_to_ccu_pipe ← gdb_msg_2
32:        end if
33:        if (msg_length = 3) then
34:            gdb_msg_3 ← ENV_to_AJIT_pipe
35:            debug_to_ccu_pipe ← gdb_msg_3
36:        end if
37:        if (PC or NPC or PSR write) then
38:            stored_PC or stored_NPC or stored_PSR ← gdb_msg_2
39:        else if (watchpoint(x) write) then
40:            stored adddress of x ← gdb_msg_2
41:        else if (read the memory / register) then
42:            ccu_data ← ccu_to_debug_pipe
43:            AJIT_to_ENV_pipe ← ccu_data
44:        else if (CONTINUE or DETACH) then
45:            debug_to_ccu_pipe ← stored_PC
46:            debug_to_ccu_pipe ← stored_NPC
47:            debug_to_ccu_pipe ← stored_PSR
48:        end if
49:        if ((set/clear the breakpoint/watchpoint) or (write memory / register)) then
50:            AJIT_to_ENV_pipe ← Acknowledge_OK
51:        end if
52:    end if
53: end function
```

2.2 Pseudo code : CCU

Algorithm 2 CCU

```
1: function CPU_CCU
2:     debug_mode ← ENABLE
3:     Initialize processor
4:     ccu_to_debug_pipe ← CONNECT_RQST
5:     while debug_msg ≠ Acknowledge_OK do          ▷ Establish connection with debugger
6:         debug_msg ← debug_to_ccu_pipe
7:     end while
8:     CCURESPONDTOGDB(GDB_CONNECTED,0)             ▷ Execute GDB commands
9:     while 1 do                                   ▷ Infinite loop
10:        ...
11:        if (error_mode) then
12:            CCURESPONDTOGDB(ERROR_MODE,0)
13:        end if
14:        ...
15:        if (trap occured) and (debug_mode = ENABLE) then
16:            CCURESPONDTOGDB(TRAP,0)
17:        end if
18:        ...
19:        pass_to_teu ← 1
20:        while pass_to_teu do
21:            ccu_to_teu ← CONTINUE
22:            teu_msg ← teu_to_ccu_pipe            ▷ wait till TEU send back a response
23:            if (breakpoint hit) and (debug_mode = ENABLE) then
24:                CCURESPONDTOGDB(BP_HIT,reg)
25:            else if (watchpoint hit) and (debug_mode = ENABLE) then
26:                CCURESPONDTOGDB(WP_HIT,reg)
27:            else
28:                pass_to_teu ← 0
29:            end if
30:        end while
31:    end while
32: end function
```

```
function CCURESPONDTOGDB(stop_reason,reg)
    debug_send_msg ← Encode (stop_reason,reg) for debugger
    ccu_to_debug_pipe ← debug_send_msg
    if ( stop_reason = breakpoint or watchpoint or trap hit) then
        ccu_to_debug_pipe ← PC
        ccu_to_debug_pipe ← NPC
        ccu_to_debug_pipe ← PSR
    end if
    debug_msg ← debug_to_ccu_pipe
    while debug_msg ≠ CONTINUE do                ▷ Execute GDB commands till CONTINUE
        Decode debug_msg
        if (msg_length = 2) then
            debug_msg_2 ← debug_to_ccu_pipe
        end if
        if (msg_length = 3) then
            debug_msg_3 ← debug_to_ccu_pipe
        end if
        if (read register x) then
            ccu_to_debug_pipe ← content of register x
        else if (write register x) then
            content of register x ← debug_msg_2
        else if (read memory) then
            mem_address ← debug_msg_2
            ccu_to_debug_pipe ← content of mem_address
        else if (write memory) then
            mem_address ← debug_msg_2
            mem_data ← debug_msg_3
            content of mem_address ← mem_data
        else if (set breakpoint/watchpoint x) then
            content of breakpoint/watchpoint register x ← debug_msg_2
        else if (remove breakpoint/watchpoint x) then
            valid bit of breakpoint/watchpoint register x ← 0
        else if (DETACH) then
            debug_mode ← DISABLE
            break
        end if
        debug_msg ← debug_to_ccu_pipe
    end while
    if (debug_msg = CONTINUE or DETACH) then
        PC ← debug_to_ccu_pipe
        NPC ← debug_to_ccu_pipe
        PSR ← debug_to_ccu_pipe
    end if
end function
```

2.3 Pseudo code : TEU

Algorithm 3 TEU

```
1:  function CPU_TEU
2:      while 1 do                                              ▷ Infinite loop
3:          while ccu_to_teu_pipe ≠ CONTINUE do                 ▷ Wait till CCU allows
4:          end while
5:          ...
6:          if (fetch_address = content of breakpoint register x) then
7:              teu_to_ccu_pipe ← Breakpoint x hit
8:              while ccu_to_teu_pipe ≠ CONTINUE do             ▷ Wait till CCU allows
9:              end while
10:         end if
11:         ...
12:         if (memory_address = content of watchpoint register x) then
13:             teu_to_ccu_pipe ← Watchpoint x hit
14:             while ccu_to_teu_pipe ≠ CONTINUE do             ▷ Wait till CCU allows
15:             end while
16:         end if
17:         ...
18:         if (trap occured) then
19:             teu_to_ccu_pipe ← trap info
20:             while ccu_to_teu_pipe ≠ CONTINUE do             ▷ Wait till CCU allows
21:             end while
22:         end if
23:     end while
24: end function
```