Using the XTA With Assembly

IN THIS DOCUMENT

When writing programs in assembly it is still possible to label code to make it portable using assembler directives.

1 Assembly Directives

The XMOS Timing Analyzer directives add timing metadata to ELF sections.

- `xtabranch` specifies a comma-separated list of locations that may be branched to from the current location.
- `xtacall` marks the current location as a function call with the specified label.
- `xtaendpoint` marks the current location as an endpoint with the specified label.
- `xtalabel` marks the current location using the specified label.
- `xtacorestart` specifies that a logical core may be initialized to start executing at the current location.
- `xtacorestop` specifies that a logical core executing the instruction at the current location will not execute any further instructions.

The `xtacall`, `xtaendpoint`, `xtalabel` directives are intended for use by the compiler only. They are used to link lines of source code with assembly instructions. All other XTA functionality provided by these directives (timing, exclusions) should be possible through the use of labels in the assembly code.

Strings used by the XTA for `xtacall`, `xtaendpoint` and `xtalabel` must not contain spaces.

2 Branch Table Example

If a branch table is written in assembly, branch target information must be added for the XTA to be able to analyze the assembly properly . This information is given in the form of a `.xtabranch` directive. For example, consider the code in Figure 1.

The XTA is not able to determine where the `bru` instruction will branch to because it is branching off a register value which is an argument to `main`. With the directive the

```
. type f, @function
. globl f
f:
    entsp 1
. xtabranch Ltarget1 , Ltarget2 , Ltarget3
    bru r0
Ltarget1 :
    bl taskA
    retsp 1
Ltarget2 :
    bl taskB
    retsp 1
Ltarget3 :
    bl taskC
    retsp 1
```

Figure 1: Setting branch targets

XTA can consider the `bru` instruction to have the three targets (`Ltarget1`, `Ltarget2`, `Ltarget3`) and the XTA can successfully time the function.

3 Core Start/Stop Example

By default the XTA, assumes that the initial logical core starts executing at the RAM base. However, if developers add another core in assembly, they also need to add `.xtacorestart` and `.xtacorestop` directives for the XTA to know that the code is reachable. For example, consider the code in Figure 2.

```
. type main , @function
. globl main
main :
    getr r1 , XS1 \ _RES \ _TYPE \ _CORE
    ldap r11 , secondCore
    init t[r1 ]:pc , r11
    start t[r1]
    ldc r1 , 0
loop :
    bf r1 , loop
    retsp 0

secondCore :
. xtacorestart
    ldc r0 , 1
    tsetmr r1 , r0
. xtacorestop
    freet
```

Figure 2: Setting core start and stop points.

With the `xtacorestart` and `xtacorestop` directives the XTA knows that the code after the label `secondCore` is reachable and hence can be analyzed.