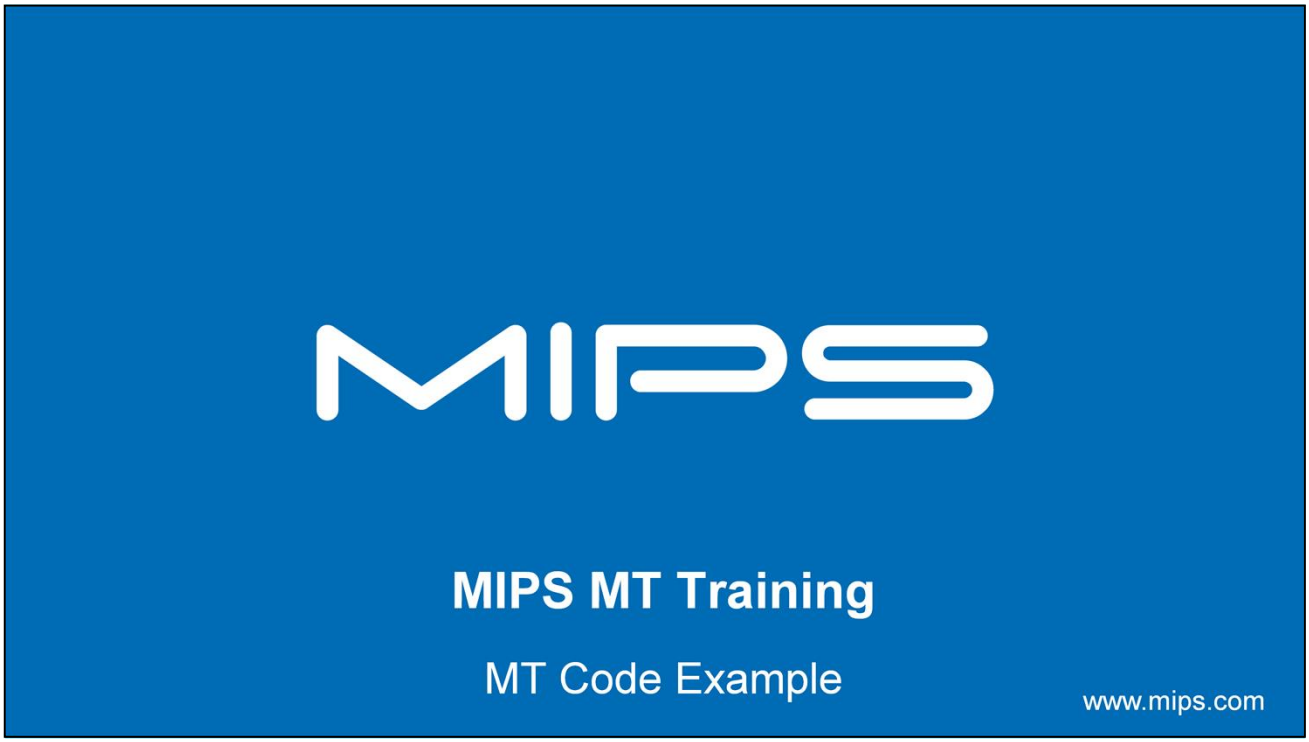

In the next 2 sections I will show you a simple example of the use of Multi threading.

MT TC Example

- **Introduction**

The following example will:

- Demonstrate the starting of three threads
- Show effects of fine grain MT

MIPS

2

+ This example show you how to start three threads.

+ you will see the effect of threading

MT TC Example

- **We will go thought the steps in the example code that create the threads and get them running.**
  - Most steps will link a view of an architectural register, with Intrinsics, macros, defines and assemble code that they produce to perform the necessary action.

MIPS

3

In this section I will do a code walk through. At each step I'll show the registers involved, the C code lines and assemble code for each step.

In the next section I will go through the compiling and running the code in the debugger.

MT TC Example

- **Intrinsic, macros and #defines for the MIPS® MT ASE**
  - Allow easy access from C code to special MT instructions and operations.
  - The Intrinsic are defined in the file include/mips/mt.h
  - #include <mips/mt.h> in your C code source file.
  - Refer to the mt.h file for more information.

I have already shown you some of the macros that allow you to program in C. Just a reminder these are located in the include/mips/mt.h file.

Turn on Virtual Processor Configuration and Disable Virtual Processing

Fields		MVPControl - CP0 #0-Sel1	Read/Write	Reset State
Name	Bits			
VPC	1	VPE Configuration State. If set, allows writing to normally read-only configuration register fields on conventional MIPS32 CPUs.	R/W	0
EVP	0	Enable Virtual Processors. If set, execute instructions for all threads on activated VPEs. If cleared, execute instructions only for thread which is running when cleared.	R/W	0

```
mips32_setmvpcontrol((mips32_getmvpcontrol() & ~MVPCONTROL_EVP) | MVPCONTROL_VPC);
Assembly code:
        mfc0 t0,c0_mvpcontrol       // read MVPControl
        li t1 2                     // load the value for the combined  VPC and EVP fields
        ins t0, t1, 0, 2            // insert VPC and EVP fields number into MVPControl register value
        mtc0 t0,c0_mvpcontrol       // write new value to the MVPControl register
        ehb                         // ensure write has completed before continuing
```

MIPS

5

The first thing the code needs to do is to put the processor into a mode where we can use the CP0 registers to configure the threads we want to run.

The MVPControl register has 2 fields, VPC and EVP.

+ Setting the VPC field will allow us to write registers that a normally are not writable on a single core MIPS processor.

+ Clearing EVP disables all multi processing so we can configure all the threads.

+ The mips32_getmvpcontrol macro reads the MVPControl register.

I clear the EVP bit using the MVPCONTROL_EVP #define

and set the VPC bit using the MVPCONTROL_VPC #define

The write is back using the mips32_setmvpcontrol macro.

+ Here is what the assemble code looks like

Setup so we can access TC1s CP0 registers with the mttr and mftr instructions

Fields		VPEControl - CP0#1-Sel1	Read/Write	Reset State
Name	Bits			
TargTC	7- 0	Target TC number to be used on MTTR and MFTR instructions	R/W	0

```
mips32_mt_settarget(TC1);
Assembly code:
        mfc0 t0,c0_vpecontrol       // read the VPEControl Register
        li t1, TC1                  // load target TC number
        ins t0,t1,0, 8              // insert TC number into VPEControl register value
        mtc0 v0,c0_vpecontrol       // write new value to VPEControl register
        ehb                         // ensure write has completed before continuing
```

MIPS

6

Assuming the thread we are executing on is thread 0, the target TC needs to be configure for thread 1. To do this use the TargTC field in the VPEControl register

Once this done the mttr - move to thread register instruction and the mftr - move from thread register instruction will be directed to Thread 1.

+ This is simple to do in C using the mips32_mt_settarget macro

+Here is the assemble code

Halt TC1

Fields		TCHalt - CP0#2-Sel4	Read/Write	Reset State
Name	Bits			
H	0	Thread Halted. If set thread has been halted and cannot be allocated, activated, or scheduled	R/W	1

```
mips32_mt_settchalt(TCHALT_H);
 Assembly code:
        li t0,1                 // load the H field
        mttc0 t0,c0_tchalt      // write the value to the TCHalt register
        ehb                     // ensure write has completed before continuing
```

MIPS

7

Before continuing, the target thread needs to be halted otherwise the change being made will be unpredictable. To make sure Thread 1 is halted before configuring it; set the H field in its TCHalt register.

+ In C I can use the mips32_mt_settchalt macro and the TCHALT_H #define

+ Here is the assemble code

Bind TC1 to VPE0

Fields		TCBind - CP0#2-Sel2	Read/ Write	Reset State
Name	Bits			
CurVPE	3 - 0	ID number of the VPE the TC is bound to	R/W	0

```
mips32_mt_settcbind (VPE0);
Assembly code:
        mttc0 zero,c0_tcbind        // write the value to the TCBind register
        ehb                         // ensure write has completed before continuing
```

MIPS

8

Bind thread 1 to VPE 0 using its TCBind register

+ The mips32_mt_settcbind macro will write the register
+ Here is the assemble code

Set Stack and Global Pointers

- **Enables the calling of C functions for this TC:**

```
unsigned int TC1_stack[4096] __attribute__((aligned(16)));
unsigned int TC1_stack_top = (unsigned int)TC1_stack + 4080;
mips32_mt_setsp(TC1_stack_top);   //  load stack pointer
```

Assembly code:

```
lw t0,-32740(gp)                  // load global variable TC1_stack_top
mttgpr t0,sp                      //  write target TC stack pointer
```

```
mips32_mt_setgp(&_gp);
```

Assembly code:

```
mttgpr gp,gp                      // move gp from gp of current thread to gp of target thread
```

MIPS

9

I now setup the stack pointer and global pointer of the thread.

I have previously allocated space for the threads stack and set the variable TC1_stack_top to the last word entry in the stack since stacks grow down.

I use the mips32_mt_setsp macro to write the stack pointer register.

+ Here is the assemble code

The Global pointer is used to reference the global variables in the small data areas. These variables are shared by all threads.

+ To set the global pointer I will use an external variable set up by the linker called _gp and the mips32_mt_setgp macro

+ Here is the assemble code notice I just copy the current threads gp register to the target thread

Set Starting function address

Fields		TCRestart - CP0#2-Sel3	Read/Write	Reset State
Name	Bits			
Restart Address	31 - 0	Address at which execution is started	R/W	0

```
mips32_mt_settcrestart(startTC1);

Assembly code:
        li t0, _startTC1            // load starting address using function lable
        mttc0 t0,c0_tcrestart       // write address to TCRestart register
        ehb                         // ensure write has completed before continuing
```

MIPS

Use the TCRestart register to tell the CPU where to start fetching instructions from for the target TC. Use the function pointer for the startTC1 function as the address to start TC from.

+ The macro mips32_mt_settcrestart sets the starting address using the startTC1 function pointer which points to the starting function for the thread.

+ Here is the assemble code

Activate TC1 and set Dynamic Allocation

Fields		TCStatus - CP0#2-Sel1	Read/ Write	Reset State
Name	Bits			
A	13	Activated. If set run instructions for this TC. Also set by FORK and cleared by YIELD $0	R/W	1
DA	15	Dynamic Allocation enable. If set TC can be allocated by FORK or de-allocated by Yield	R/W	0

```
mips32_mt_settcstatus(mips32_mt_gettcstatus() | (TCSTATUS_A | TCSTATUS_DA) );
Assembly code:
        mftc0 v0,c0_tcstatus        // read the TCStatus register
        ori v0,v0,0xa000            // or in the A and AD bits
        mttc0 v0,c0_tcstatus        // write the TCStatus register
        ehb                         // ensure write has completed before continuing
```

MIPS

11

Next activate the thread and make it available for use with Fork and Yield instructions using the TCStatus register.

+ To active the thread, set the A field (activated)

+ To make the thread Yieldable it must be marked a Dynamically Allocatable, set the DA bit

Note: This example does not use the fork instruction but it will use the Yield instruction at the end of execution so we do need to enable Dynamic Thread allocation by setting the DA field.

+ I use the mips32_mt_gettcstatus to get the curent value and the TCSTATUS_A and TCSTATUS_DA #define to set the bits and the mips32_mt_settcstatus macro to write the value to the register.

+ Here is the assemble code

MIPS® MT TCHalt - CP0#2-Sel4

Fields		TCHalt - CP0#2-Sel4	Read/Write	Reset State
Name	Bits			
H	0	Thread Halted. If set thread has been halted and cannot be located, activated, or scheduled	R/W	1

```
mips32_mt_settchalt(0);
Assembly code:
        mttc0 zero,c0_tchalt        // only bit in register move the value in the zero register to TCHalt
        ehb                         // ensure write has completed before continuing
```

MIPS

12

The last step in configuring the Thread is to un-halt it. By doing this the thread can be scheduled and instructions can be fetched once I enable multi threading. Clearing the H bit in the TCHalt register un-halts the thread.

+ The mips32_mt_settchalt macro with a zero argument will clear the H bit in the TCHalt register.

+ Here is the assemble code

MT TC Example

- **Set up TC2 just like TC1**

  mips32_mt_settarget(TC2)

  - The rest is the same as for TC1
    - NOTE: You need to use different values for stack and starting function. Use TC2_stack_top for stack.

To setup thread 2 I just set the Target TC to 2 and then set it up the same as thread 1.

+ The one thing that must be different for each thread is the stack other wise the stack will be corrupted. You can use the same starting code address since each thread will have its own stack and therefore each will have its own context. However this example will use slightly different code for each thread.

MT TC Example Enable Threading

Fields		VPEControl - CP0#1-Sel1	Read/Write	Reset State
Name	Bits			
TE	15	Thread Enable. If unset only one TC may execute.	R/W	0

```
mips32_mt_setvpecontrol(mips32_mt_getvpecontrol() | VPECONTROL_TE);
Assembly code:
        mftc0 v0,c0_vpecontrol      // read in the VPEControl register
        ori v0,v0,0x8000            // set the TE bit
        mttc0 v0,c0_vpecontrol      // write the VPEControl register
        ehb                         // ensure write has completed before continuing
```

MIPS

14

After I have initialized all the threads I need to enable threading on the VPE. I do this by setting the TE bit in the VPEControl register.

+ To do this I use the mips32_mt_getvpecontrol to get the current value of the VPEControl register then I use the VPECONTROL_TE #define to set the TE bit and the mips32_mt_setvpecontrol to write it back.

+ here is the assemble code

Turn off configuration flag and enable Virtual Processing

- MVPControl - CP0#0-Sel1

Fields		MVPControl - CP0 #0-Sel1	Read/Write	Reset State
Name	Bits			
VPC	1	VPE Configuration State. If set, allows writing to normally read-only configuration register fields on conventional MIPS32 CPUs.	R/W	0
EVP	0	Enable Virtual Processors. If set, execute instructions for all threads on activated VPEs. If cleared, execute instructions only for thread which is running when cleared.	R/W	0

```
mips32_setmvpcontrol((mips32_getmvpcontrol() & ~MVPCONTROL_VPC) | MVPCONTROL_EVP);
Assembly code:
        mfc0 t0,c0_mvpcontrol       // read MVPControl
        li t1 1                     // load the value for the combined  VPC and EVP fields
        ins t0, t1, 0, 2            // insert VPC and EVP fields number into MVPControl register value
        mtc0 t0,c0_mvpcontrol       // write new value to the MVPControl register
        ehb                         // ensure write has completed before continuing
```

MIPS

15

Last, to finally enable Multi threading and start all enabled threads executing I need to turn off configuration mode and Enable Virtual Processing. These are set in the MVPControl register.

+ I use the mips32_getmvpcontrol macro to read the register

Then the MVPCONTROL_VPC #define to clear the VPC bit to turn off the configuration state

and the MVPCONTROL_EVP #define to set the EVP bit to enable virtual processing.

Then write the register using the mips32_setmvpcontrol macro.

+ Here is the assemble code

MT TC Example

- **Start count function on TC0**

```
count(0);

return (0); // Never gets here.
```

MIPS

16

At this point all threads will be scheduled and will start running code.

Now put the current thread that executed the initialization code, thread 0 into the mix by calling the count function.

Note; the code will never execute the return call because all threads will be yielded including thread 0.

MT TC Example

- **Count Functions**
  - Two simple functions
    - Both increment a counter element in the same array using the argument given, which is the TC number.
    - Count function – increments array element by using a cached address in KSEG0. When the counter reaches 2000 the TC will yield (de-allocate)
    - Nccount function– increments array element by using a Uncached address in KSEG1. This function will stall waiting to read the count value.

MIPS 17

The rest of the code will be used to show multi threading and how the threads behave when run from cache or straight out of ram.

+ To do this there are 2 simple functions each of which will increment a counter in a global array.

+ The first function Count, access the counter array using a cached address.

+ the second function Ncount increments a counter in the same array but through a uncached address.

MT TC Example

- **Count Functions (continued)**
  - Thread behavior:
    - The threads that use the count function will not stall. The count will increase quicker than the thread that uses the Nccount function, because that function stalls.
    - If these were threads in a non-MT system, nothing would execute when the Nccount function stalls.
    - This example shows how MT fine-grain threading allows other threads to execute while other threads are stalled waiting for memory.

MIPS

18

What you will see is the threads that use the cached address to increment their counter will not stall and these threads will execute more then the thread that is writing to the uncached address. This is because the threads using the cached address will be allowed to execute while the thread that uses the non cached address stalls waiting for the load to complete.

Fine grain multi threading allows other threads to execute while another thread is stalled. Normally stall cycles would be wasted in a non threaded CPU.

MT TC Example

- **Count Functions (continued)**
  - Yield behavior:
    - To show how to use the Yield command, the threads that use the count function will Yield after the count reaches a certain point.
    - The thread that used the Nccount function will then run on alone and we will see its count progress.

The thread shows the effect of the Yield command.

+ Each thread will use the Yield command to terminate itself once terminal count has been reached.

+ You'll see that the threads using the count function will terminate before the thread executing the Ncount function because they have gotten to run while the Ncount function was stalled.