

GTE Programming Guide

July 19, 1996

Version 1.0

Sony Computer Entertainment Inc.

Confidential

Format

GTE is a vector/matrix processor implemented as "coprocessor 2" under the MIPS architecture specification. The data format it handles consists of fixed decimal (fractional) real numbers.

Registers

Coprocessors in the MIPS architecture can have two sets of registers: "data registers" and "control registers." GTE has 32 data registers and 32 control registers. These are all 32-bit registers, and access by the CPU is performed in register units. However, some registers are divided into multiple 8- or 16-bit fields. GTE references / changes only GTE registers when it is performing calculations.

For a detailed description of the various registers, refer to "GTE Register Specification."

Register access instructions:

Data is transferred between GTE registers and CPU registers, or GTE registers and main memory (including the scratchpad) by executing the CPU instructions listed below.

| CPU instruction | Source | Destination |
|-----------------|------------------------------|------------------------------|
| lwc2 | Memory scratchpad | GTE data register |
| swc2 | GTE data register | Memory scratchpad |
| mtc2 | CPU general-purpose register | GTE data register |
| mfc2 | GTE data register | CPU general-purpose register |
| ctc2 | CPU general-purpose register | GTE control register |
| cfc2 | GTE control register | CPU general-purpose register |

The instructions mtc2, mfc2, ctc2, and cfc2 transfer data between registers. However, as is the case for on-cache memory accesses, delayed loads should be used. In examples such as those listed below, nop should be inserted into the delay slot.

Example 1:

```
cfc2    v0,C2_FLAG
and     v0,v0,v1 # No good
```

Example 2:

```
cfc2    v0,C2_FLAG
nop
and     v0,v0,v1      # delay slot
```

Register names:

The macro definition of GTE register names can be found in the include header file "gtereg.h," which is part of the PlayStation library. The names of the macros are formed by adding the prefix "C2_" to the register names used in the "GTE Register Specification."

| Register number | Data register | Control register |
|-----------------|---------------|------------------|
| 0 | C2_VXY0 | C2_R11R12 |
| 1 | C2_VZ0 | C2_R13R21 |
| 2 | C2_VXY1 | C2_R22R23 |
| 3 | C2_VZ1 | C2_R31R32 |
| 4 | C2_VXY2 | C2_R33 |
| 5 | C2_VZ2 | C2_TRX |
| 6 | C2_RGB | C2_TRY |
| 7 | C2_OTZ | C2_TRZ |
| 8 | C2_IR0 | C2_L11L12 |
| 9 | C2_IR1 | C2_L13L21 |
| 10 | C2_IR2 | C2_L22L23 |
| 11 | C2_IR3 | C2_L31L32 |
| 12 | C2_SXY0 | C2_L33 |
| 13 | C2_SXY1 | C2_RBK |
| 14 | C2_SXY2 | C2_GBK |
| 15 | C2_SXYP | C2_BBK |
| 16 | C2_SZ0 | C2_LR1LR2 |
| 17 | C2_SZ1 | C2_LR3LG1 |
| 18 | C2_SZ2 | C2_LG2LG3 |
| 19 | C2_SZ3 | C2_LB1LB2 |
| 20 | C2_RGB0 | C2_LB3 |
| 21 | C2_RGB1 | C2_RFC |
| 22 | C2_RGB2 | C2_GFC |
| 23 | Undefined | C2_BFC |
| 24 | C2_MAC0 | C2_OFX |
| 25 | C2_MAC1 | C2_OFY |
| 26 | C2_MAC2 | C2_H |
| 27 | C2_MAC3 | C2_DQA |
| 28 | C2_IRGB | C2_DQB |
| 29 | C2_ORGB | C2_ZSF3 |
| 30 | C2_LZCS | C2_ZSF4 |
| 31 | C2_LZCR | C2_FLAG |

Commands

GTE can perform an entire series of calculations essential for graphics programming (such as coordinate transformation, perspective transformation, and light source calculation) by executing a single command. Also, general-purpose matrix and vector calculations (such as matrix calculation, outer product, and interpolation) are available as commands. In all of the above cases, the calculation speed is several times faster than if the calculations in question were performed by the CPU. Refer to the "GTE Command Reference" for a detailed description of the available GTE commands. Also, GTE commands are macro defined in the file "inline.h," which is included with DMPSX.

Delay slots:

CPU instructions that execute coprocessor 2 commands (referred to as "cop2"), require two delay slots for preceding GTE-related instructions.

Example 3:

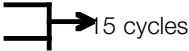
```
mtc2    v0,C2_VXY0
nop                      # delay slot
nop                      # delay slot
RTPS
```

Command execution cycles:

The various GTE commands require the number of cycles listed below to complete. After a coprocessor 2 execute instruction is issued, if the prescribed number of cycles is not left open, and either a GTE register read instruction (swc2, mfc2, cfc2) or another coprocessor 2 command execute instruction is issued, the CPU will stall until the initial coprocessor 2 instruction has completed execution.

Example 4:

```
RTPS
## interlock
cfc2    v0,C2_FLAG
```



Example 5:

```
RTPS
add     v1,v2,v3
sub     v1,v2,v3
## interlock
cfc2    v0,C2_FLAG
```



Coding limitations:

- [1]A GTE instruction (cop2) must not be executed in an event handler or callback function.
- [2]A GTE instruction (cop2) must not be inserted into the delay slot following a jump or branch instruction.
- [3]A GTE register access instruction (lwc2, swc2, mtc2, mfc2, ctc2, cfc2) must not be inserted into the delay slot following a jump or branch command.
- [4]A GTE register load instruction (lwc2, mtc2, ctc2) must not be used between a GTE instruction (cop2) and a GTE register save instruction (swc2, mfc2, cfc2) or between a GTE instruction (cop2) and another GTE instruction (cop2).

Example 6:

```
/* cop2-load-save (NG) */
RTPS
...
mtc2    v0,C2_VXY0
...
cfc2    v0,C2_FLAG
```

```
/* cop2 */
/* cpu instructions */
/* NG !!!!! */
/* cpu instructions */
/* save instruction */
```

Example 7:

```
/* cop2-load-cop2 (NG) */
RTPS
...
mtc2    v0,C2_VXY0
...
NCLIP
```

```
/* cop2 */
/* cpu instructions */
/* NG !!!!! */
/* cpu instructions */
/* cop2 */
```

- [5]If a GTE register to which data is to be loaded is not being referenced or overwritten by a GTE command that is currently executing, it is possible to execute a command which transfers data to the GTE register without worrying about the GTE

command (cop2).

Example 8:

```
/* cop2-load-save (OK) */
RTPS
...
mtc2    v0,C2_VXY1
...
cfc2    v0,C2_FLAG
/* cop2 */
/* cpu instructions */
/* OK !! */
/* cpu instructions */
/* save instruction */
```

Example 9:

```
/* cop2-load-cop2 (OK) */
RTPT
...
mtc2    v0,C2_RGB
...
NCLIP
/* cop2 */
/* cpu instructions */
/* OK !! */
/* cpu instructions */
/* cop2 */
```

Recommended development style

When coding in assembler, programs that violate some of the above rules may appear to run properly at first glance. However, such violations tend to become evident as bugs that are extremely difficult to track down, such as incorrect operation during an interrupt. For this reason, programmers are advised to avoid coding directly in assembler as much as possible.

To prevent bad code from being generated, the following development sequence should be used.

C (libgte) -> C (DMPSX) -> Assembler (DMPSX)