Integrated Device Technology, Inc.

EMBEDDING ASSEMBLY INSTRUCTIONS INSIDE C-SOURCE CODE

TECHNICAL NOTE TN-18

By Sugan Subramanian

INTRODUCTION

This is a tech note on how to embed assembly instructions inside C source code. It is targeted towards programmers who have some knowledge of C-language and R3000 assembly language.

In IDT/C™ 5.0, assembly instructions can be inlined inside any genuine block of C-code. A genuine block of C-code is a section of C-code enclosed by open and closed curly braces. The inlined assembly may include synthetic assembly instructions. These instructions are expanded during compile/assembly phase of the compiler. The format agreed by the IDT/C 5.0 compiler depends on whether or not the inlined assembly lines require arguments, and whether these arguments are read, written, or both.

Specifically there are 4 cases to consider:

- inline without any parameters
- inline with read only parameters
- inline with write only parameters
- inline with read and write parameters

These four cases will be discussed elaborately in the following sections.

INLINE ASSEMBLY LINES WITHOUT ANY PARAMETERS

Format:

asm("<asm instrct1> ; <asm istrct2> ; <asm instrct2>;
... <asm instrctn>")

e.g:

```
unsigned int get_addr()
{
asm("li $2,0x80020000 ; lui $3, 0");
}
```

Description:

1. "get_addr" is a function that takes no arguments and returns an unsigned integer.
2. The inlined portion of the function body computes the return value == (0x80020000) that is saved at $2 (or) v0 and initializes $3 (or) v1 with zero.

Constraints:

All assembly instructions including synthetic instructions are allowed.

All register names should have hardware mnemonics.
i.e.:

General registers are $0, $1, .. , $31
Coprocessor 0 registers (has TLB, configuration specific registers) are $0,$1, ... , $31
Coprocessor 1 registers (has Floating Point Accelerator specific registers) are $0, $1, ... , $31
Coprocessor 2 registers are $0, $1, ... , $31
Coprocessor 3 registers are $0, $1, ... , $31

INLINE ASSEMBLY LINES WHICH USE WRITE-ONLY PARAMETERS

Format:

asm("<asm instrct1> ; <asm istrct2> ; <asm instrct2>; ...
<asm instrctn>"
: "=<write-only_param1 format>" (<write-only_param1 name>),
"=<write-only_param2 format>" (<write-only_param2 name>),
... "=<write-only_paramk format>" (<write-only_paramk name>));

e.g:

```
int i,j;
void main()
{
Intialize_Globals();
printf("value of i = %d and j = %d\n",i,j);
}
void Initialize_Globals()
{
asm("ori %0,$0,3 ; ori %1, $0, 4"
: "=r" (i), "=r" (j));
}
```

Description:

1. "Initialize_Globals" is a function that takes no arguments and returns nothing.
2. The inlined portion of the function body initializes the global variables "i" and "j". Uses "i" and "j" as write-only parameters. Parameter "i" is referenced by %0 and "j" is referenced by %1. "=r" is the format for both "i" and "j". "=r" specifies that the following write-only parameter has a general register associated with it.

Constraints:

All assembly instructions including synthetic instructions are allowed.

All register names should have hardware mnemonics.
In R3000, the following are the possible hardware mnemonic:
General registers are $0, $1, .. , $31
(has TLB, configuration specific registers)

Coprocessor 0 registers are $0, $1, ... , $31
(has Floating Point Accelerator specific registers)
Coprocessor 1 registers are $0, $1, ... , $31
Coprocessor 2 registers are $0, $1, ... , $31
Coprocessor 3 registers are $0, $1, ... , $31

Write-only parameters can be either global or local variables. Write-only parameters are indexed from 0 to n-1, where n is the number of parameters used in the inlined code. Inside the inlined code, write_only_parameter1 is accessed by %0, write_only_parameter2 is accessed by %1, and so on. These are the formats that are allowed for write-only parameters:

"=r" ___ Specifies that the write-only parameter has a general register assigned to it.
"=f" ___ Specifies that the write-only parameter has a floating point register assigned to it.

INLINE ASSEMBLY LINES WHICH USES READ-ONLY PARAMETERS

Format: asm("<asm instrct1> ; <asm istrct2> ; <asm instrct2>; ... <asm instrctn>"
:: "<read-only_param1 format>" (<read-only_param1 name>),
"<read-only_param2 format>" (<read-only_param2 name>),
..."<read-only_paramk format>" (<read-only_paramk name>));

e.g:

```
void main()
{
print("INLINE VAL = %d\n", return_3());
}

int return_3()
{
asm("ori $2,$0,%0 ; ori $3, $0, %1"
:: "n" (3), "n" (4));
}
```

Description:

1."return_3" is a function that takes no arguments and returns integer value 3.
2. The inlined portion of the function computes the return value.
3. Whenever we use read only parameters without write only parameters, we have to use two colons "::" preceding them to specify that there are no write only parameters.

Constraints:

All assembly instructions including synthetic instructions are allowed.
All register names should have hardware mnemonics.
i.e.
General registers are $0, $1, .. , $31
Coprocessor 0 registers (has TLB, configuration specific registers) are $0, $1, ... , $31
Coprocessor 1 registers (has Floating Point Accelerator specific registers) are $0, $1, ... , $31
Coprocessor 2 registers are $0, $1, ... , $31
Coprocessor 3 registers are $0, $1, ... , $31

Read-only parameters are indexed from 0 to n-1, where n is the number of parameters used in the inlined code. Inside the inlined code, Read-only_parameter1 is accessed by %0, Read-only_parameter2 is accessed by %1, and so on. These are the formats that are allowed for read-only parameters:

"r" ___ Specifies that the parameter has a general register assigned to it.
"f" ___ Specifies that the parameter has a floating point register assigned to it.
"n" ___ Specifies that the parameter is an immediate value.
"m"___ Specifies that the parameter is a memory address.
"o" ___ Specifies that the parameter is an offsettable memory address.
"X" ___ Specifies that the parameter can be any of the above.

INLINE ASSEMBLY LINES THAT USES WRITE-ONLY AND READ-ONLY PARAMETERS

Format:

asm("<asm instrct1> ; <asm istrct2> ; <asm instrct2>; ... <asm instrctn>"
: "=<output_var1 format>" (<output_var1 name>),
"=<output_var2 format>" (<output_var2 name>),
... "=<output_vark format>" (<output_vark name>)
: "<input_var1 format>" (<input_var1 name>),
"<input_var2 format>" (<input_var2 name>),
..."<input_vark format>" (<input_vark name>));

e.g:

```
#define ARRAY_SIZE_IN_BYTES 40
int b[20];
void main()
{
int a[10];
int i,j,k;
{asm (
"
.set    noreorder
   li      $11,%2;
   addiu   %0,%3;
1:;
   sw      $11,0(%0);
   addiu  $11,-4;
   bnez    $11,1b;
   addiu   %0,-4;
   li      %1,%4;
.set    reorder"
```

```
:: "r" (a), "r" (j), "n"
(ARRAY_SIZE_IN_BYTES),
   "n" (10*4), "m" (b)
:  "$11");}

printf ("return val = %d\n",j);
i=-1;

while (++i < 10)
   printf("a[%d] = %d\n",i,a[i]);
}
```

Description:

1. This program initializes an integer array of 10 with values starting from 0 through 36 by an increment of 4 and displays the array.
2. The inlined portion not only initializes the array but demonstrates one peculiar inline feature, how to use read-write parameter.
3. We are allowed to use registers inside inlined assembly lines as long as we declare that they will be clobbered. This is done by giving register name(s) preceded by three colons (":::") if there are no write-only and read-only parameters, a colon (":") following the read only parameter(s) if there are read-only parameters, and two colons ("::") following the write only parameter(s) if there are only write-only parameters.

Constraints:

All assembly instructions including synthetic instructions are allowed.

All register names should have hardware mnemonics. i.e.

General registers are $0, $1, .. , $31
Coprocessor 0 registers (has TLB, configuration specific registers) are $0, $1, ... , $31
Coprocessor 1 registers (has Floating Point Accelerator specific registers) are $0, $1, ... , $31
Coprocessor 2 registers are $0, $1, ... , $31
Coprocessor 3 registers are $0, $1, ... , $31

Whenever a parameter is used for reading and writing, declare such parameters to be either read-only or write-only and not both. This convention eliminates a lot of confusion. In the previous example, parameter "j" and "a" are declared to be read-only and used for both reading and writing. It is appropriate because both "j" and "a" are of type "r" (have general registers associated with them). Only read-only parameters that have registers associated with them are writable.

GENERAL RULES WHILE INLINING ASSEMBLY LINES

Always enclose your inlined assembly lines by a block of ".set noreorder" and ".set reorder" directives so that compiler leaves the inlined assembly lines untouched even if the entire code is optimized. However, some harmless warning messages are generated by the assembler (IDT/C 5.0) when the synthetic assembly instructions are expanded; they can simply be ignored.

Declare all your variables that are read from and the immediate values that are used inside inlined assembly to be read-only parameters. Declare all variables that are written to as write-only parameters. Whenever a temporary register is used inside inlined assembly code always make sure it gets declared as clobbered.

SUMMARY

Inlining assembly lines inside of c-code is a boon in itself if the only way of optimizing your c-code is through having different sections of it in assembly. In IDT/C 5.0, inlining assembly lines is complemented by the ability to use local and global variable names as aliases to the registers assigned to them.