ARM GCC inline assembly

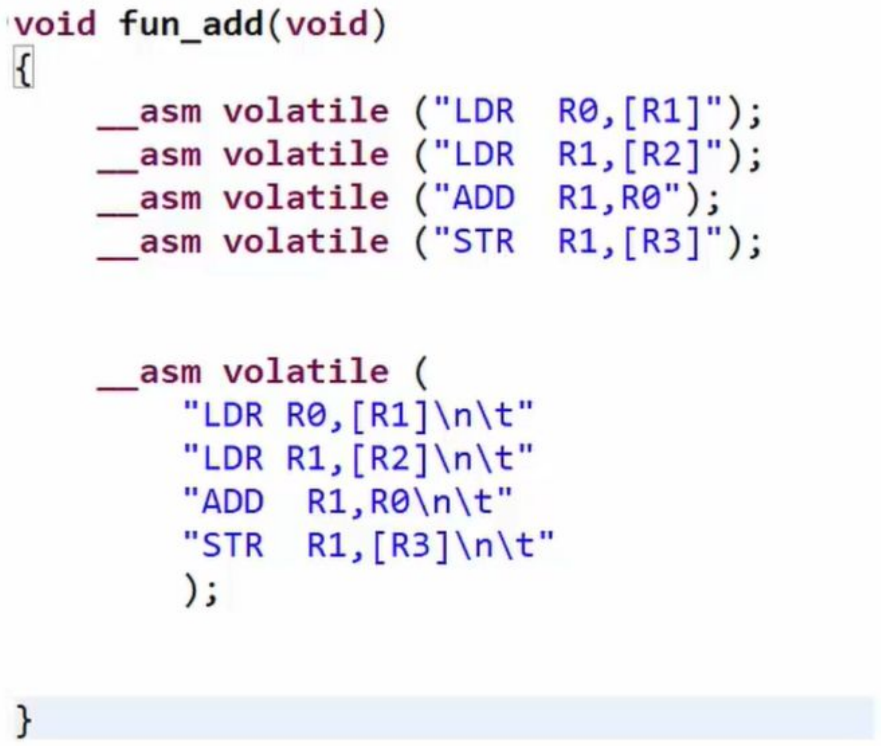
ARM GCC inline assembly code

- Inline assembly code is used to write pure assembly code inside a ‘C’ program.
- GCC inline assembly code syntax shown below

  Assembly instruction : MOV R0,R1

  Inline assembly statement : __asm volatile(“MOV R0,R1”);

- LDR R0,[R1]
- LDR R1,[R2]
- ADD R1,R0
- STR R1,[R3]



```
void fun_add(void)
{
    __asm volatile ("LDR  R0,[R1]");
    __asm volatile ("LDR  R1,[R2]");
    __asm volatile ("ADD  R1,R0");
    __asm volatile ("STR  R1,[R3]");


    __asm volatile (
        "LDR R0,[R1]\n\t"
        "LDR R1,[R2]\n\t"
        "ADD  R1,R0\n\t"
        "STR  R1,[R3]\n\t"
        );


}
```

‘C’ variable and inline assembly

- Move the content of ‘C’ variable ‘data’ to ARM register R0.
- Move the contents of the CONTROL register to the ‘C’ variable “control_reg”.

General form of an inline assembler statement

Volatile attribute to the asm statement to instruct the compiler not to optimize the assembler code

Clobber list is mainly used to tell the compiler about modifications done by the assembler code

```
__asm volatile (code : output operand list : input operand list : clobber list);
```

Assembly mnemonic defined as a single string

A list of output operands, separated by commas.

A list of input operands, separated by commas.

General form of an inline assembler statement

```
__asm volatile (code : output operand list : input operand list : clobber list);
```

```
__asm volatile(“MOV R0,R1”);
```

(code)

```
__asm volatile(“MOV R0,R1”: : :);
```

(code)

Exercise

Load 2 values from memory , add them and store the result back to the memory using inline assembly statements.

Input/output operands and Constraint string

Each input and output operand is described by a constraint string followed by a C expression in parenthesis.

Input/Output Operand Format :

“<Constraint string>” (< ‘C’ expression>)

Constraint string = constraint character + constraint modifier

Example 1 : Move the content of ‘C’ variable ‘val’ to ARM register R0

Instruction ⇒ MOV

Source ⇒ a ‘C’ variable ‘val’ (INPUT)

Destination ⇒ R0 (ARM core register)

```
__asm volatile (“MOV R0,%0”: : “r”(val) );
```

Thank You