ASM MEMENTO

This memento summarizes the use of the inline assembly feature provided by GCC. An inline assembler is a feature of some compilers that allows low-level code written in assembly language to be embedded within a program, among code that otherwise has been compiled from a high-level language (such as C).

A full set of ASM instructions can be found in [Intel 10]. A good start is consulting [San 03] and [Rao 01], and the asm codes available on Moodle.

First asm example: simple assignment (in bold italic: pure assembly code)

```
main() { // assignment: m = n;
  int n=10, m;

  asm(
    "movl %1, %0" // move %1 (n) to %0 (m), in long-word (32-bit) memory reference

    : "=r" (m)  // output list: m is accessible via %0 in write-only mode
    : "r" (n)   // input list: n is accessible via %1
  );
}
```

Second asm example: simple loop (in bold italic: pure assembly code)

```
main() { // while(++counter != max){};
  int counter=0, max=100;
  asm(
  "loop:          \n\t"  // declare the label 'loop'
    "incl %0      \n\t"  // increment %0 (counter), in long (32-bit) memory reference
    "cmp  %0, %1  \n\t"  // compare %0 (counter) and %1 (max), and
                         // set the status register to 'equal' or 'not equal'
    "JNE loop     \n\t"  // jump to label 'loop' if status register is set to 'not equal'
    : "+m" (counter)  // output operand: counter is accessed in read and write mode via %0
    : "r" (max)       // input operand: max is accessible in read mode via %1
  );
}
```

Some ASM instructions and AT&T syntax

dec arg	decrement the value of arg by one
inc arg	increment the value of arg by one
mov src, dest	move the value from src to dest
cmp op1, op2	compare the values and set the status register to ‘equal’ or ‘not equal’
jmp label	jump to ‘label’
jne label	JNE (Jump Not Equal) : jump to ‘label’ if the status register is set to 'not equal'

xchg op1, op2	atomic operation (i.e. no interruption by scheduler): exchange the two values op1 and op2
lock	next instruction is locked (assert LOCK# signal for duration of the next instruction, i.e. add atomicity to the next instruction)
cli	Clear interrupt flag (i.e. clear the IF flag in the EFLAGS register); maskable external interrupts are disabled
sti	Set interrupt flag; external maskable interrupts are enabled at the end of the next instruction. (This behavior allows external interrupts to be disabled at the beginning of a procedure and enabled again at the end of the procedure.)

Op-code suffixes of ’b’, ’w’, ’l’ and ‘q’ specify resp. byte (8-bit), word (16-bit), longword (32-bit) and quad-word (64-bit) memory references; e.g. ‘movb’ moves 8-bit.

An integer operand (one with constant value) is preceded by ’$’, e.g. $3 for the value 3.

Register names are prefixed by %, e.g. if eax (the EXTENDED ACCUMULATOR REGISTER) is to be used, write %eax (and if its value is used write %%eax).

ASM inline

The basic ASM inline format is very simple:

```
asm(assembler template);
```

where the assembler template has one of the following format:

```
" instr1 ;  instr2 "
```

or

```
" instr1 \n\t "
" instr2 "
```

In the first case the entire set of assembly instructions is enclosed within double quotes and the assembly instructions are separated by a semicolon. In the second case each assembly instruction is enclosed within double quotes and the assembly instructions are separated by a newline and a tab (hint: gcc sends each item inclosed within double quotes as a string to the asssembler; so by adding the newline and tab the correct formatted lines are sent to the assembler).

If the keyword 'asm' conflicts with something in the program, one can use the format :

```
__asm__("assembler template ");
```

If the assembly instruction must execute where it has been put (e.g. must not be moved out of a loop as an optimization) add the keyword 'volatile' (or __volatile__) :

```
asm volatile ("assembler template");
```

Extended ASM inline

The extended inline format allows to specify the input, output and clobbered registers:

```
asm(assembler template
      : list of the output register operands
      : list of the input register operands
      : list of clobbered registers
);
```

In the assembler template, each input/output register operand is referenced by a number prefixed by %, e.g. %3. The first operand in the output list is numbered 0, the second 1, and so on.

Register operand constraint

"r"	gcc keeps the variable in any of the available General Purpose Registers (GPRs).
"m"	gcc keeps the variable in memory.

Constraint Modifiers (to be put in first place)

"="	Means that the operand is write-only by the assembler template.
"+"	Means that the operand is both read and written by the assembler template.

Clobber Options

Sometimes an instruction knocks out certain specific registers. The most common example of this is a function call, where the called function is allowed to do whatever it likes with some registers.

The clobber list is a list of all the registers that are changed by the template but not listed in the input and output operands. We have to specify this list such that gcc will not assume that the values it loads into these registers will be valid. There are moreover two special cases for clobbered values: "memory" and "cc" :

"%eax", “%ebx”, …	The clobbered register %eax tells GCC that the value of %eax will be used inside "asm". GCC will not use a clobbered register for inputs or outputs. Do not use a clobbered register for a temporary register, but let gcc choose the most convenient register by making up a dummy output with “=r”. 64-bit registers are named rax, rbx, …, and 32-bit registers eax, ebx, …
"memory"	Add “memory” if assembly instructions write to some memory (other than a listed output) and GCC shouldn’t cache memory values in registers across this asm. An asm memcpy() implementation would need this. Add also the volatile keyword (as asm postfix) if the memory affected is not listed in the input or output registers. You do *not* need to list “memory” just because outputs are in memory; gcc understands that.
"cc"	Add “cc” if assembly instructions can alter the condition codes (i.e. the bits in the flags register jump or other operators may look at). It is not necessary on all machines, but it is always legal to be specified; e.g. on x86 architecture it is not necessary.

Note that with the option –S you can consult the assembly code generated by gcc :

```
gcc -S -fverbose-asm your_file.c
```

Further Readings

Miscellaneous OS and System Architecture Tutorials (inclusive spinlocks, etc.)

[Bon 10] Bona Fide: “OS Developer”; http://www.osdever.net/, see also http://www.osdever.net/tutorials.php?cat=0&sort=1

NASM (Netwide ASM)

[Swa 02] Derick Swanapoel: “NASM (Netwide ASM): Linux Assembly Tutorial”, http://www.cin.ufpe.br/~if817/arquivos/asmtut/index.html

x86 Assembly

[Col 00] Clark Coleman: “Using Inline Assembly with gcc”, http://www.osdever.net/tutorials/gccasmtut.php?the_id=68

[Wikipedia 10] Wikipedia : “X86 Assembly language”; http://en.wikipedia.org/wiki/X86_assembly_language

[WikiBook 10a] WikiBooks : “X86 Assembly/X86 Instructions”; http://en.wikibooks.org/wiki/X86_Assembly

x86 General Purpose Register

[WikiBook 10b] WikiBooks : “X86 Assembly/X86 Architecture”; http://en.wikibooks.org/wiki/X86_Assembly/X86_Architecture

x86 Instructions

[WikiBooks 10c] WikiBooks : “X86 Assembly/X86 Instructions”; http://en.wikibooks.org/wiki/X86_Assembly/X86_Instructions

References

[Intel 10] “Intel Pentium Instruction Set Reference (Basic Architecture Overview)”; http://faydoc.tripod.com/cpu/ (last visit 20 April 2010) (if you know the opcode name, just suffix this URL with ‘name.htm’, e.g. dec.htm)

[San 03] S. Sandeep: GCC-Inline-Assembly- HOWTO; http://www.ibiblio.org/gferg/ldp/GCC-Inline-Assembly-HOWTO.html (last visit 20 April 2010)

[Rao 01] Bharata B. Rao : “Inline assembly for x86 in Linux”; IBM Software Labs, India, March 2001, http://www.ibm.com/developerworks/linux/library/l-ia.html

Mario Sitz, Béat Hirsbrunner and Ammar Halabi, University of Fribourg, Switzerland
Version 0.2: 4 April 2009 (MS); Version 1.0: 20 April 2010 (BH), rev. 30 Nov. 2013 (BH, AH), rev. May 30 2017 (PCM).