750 Naples Street • San Francisco, CA 94112 • (415) 584-6360 • http://www.pumpkininc.com

Understanding Changes in Salvo Code Size for Different PICmicro Devices

Introduction

When compiling a Salvo™ multitasking application for different Microchip PICmicro devices, you may notice a sizeable difference in the code size (ROM). This Application Note lends some insight into this matter.

Same Program, Different ROM Requirements

Below are the results of compiling the same Salvo multitasking application[1] on a range of similar PICmicro devices using the HI-TECH PICC compiler:

device	library used[2]	ROM (words)	RAM (bytes)
PIC16F84	FPI400.lib	372	33
PIC16C64	FPI401.lib	367	33
PIC16C717	FPI402.lib	404	33
PIC16C65	FPI411.lib	401	33
PIC16C773	FPI412.lib	438	33
PIC16C77	FPI422.lib	472	33

The devices range from the PIC16F84, with only 1K instruction words of program memory (ROM) and 68 bytes of RAM in 1 bank, to the PIC16C77, with 8K of ROM and 368 bytes of RAM in 4 banks. It's immediately apparent that as the number of ROM and RAM banks per PICmicro device rises, so does the code size.[3]

Why the Variation?

PICmicro devices are RISC-based, Harvard architecture microcontrollers with short instruction word lengths. In order to

All trademarks mentioned herein are properties of their respective companies.

explicitly change the program counter or access file registers, the desired ROM or RAM address must be contained within the instruction word. Since the instruction word length is relatively small (14 bits for the mid-range PIC16 devices and 16 bits for the high-end PIC17 devices), and since several bits must be reserved for the opcode itself, there isn't much room left over for explicit address bits.

Consequently, the PICmicro devices employ a means of page and bank switching to access memory over an address range beyond what a single instruction can hold. In the PIC16 devices, up to 11 bits of the 14-bit instruction word can be dedicated to an absolute code address (say, for a `GOTO` instruction), and up to 7 bits can be dedicated when operating on file registers in RAM. That means that in order to access more than 2K of ROM, or more than 128 bytes of RAM, a scheme involving "some extra bits" is required.

The PICmicro devices employ a `PCLATH` (Program Counter Latch High) register which, when combined with 11 dedicated instruction address bits, allow a `CALL` or `GOTO` anywhere in the available address space. They also employ register page bits (`RPn`) to select a particular RAM bank to operate on.

PICC (and other compilers) manage memory accesses behind the scenes and make it transparent to the programmer. It may be possible to control some memory issues (e.g. in which RAM bank variables should be located), but others (e.g. where a particular function is located in ROM) are often handled automatically by the compiler.

The Role of the Compiler

While it is beyond the scope of this Application Note to discuss the code generated by the compiler for the figures above, suffice it to say that with a larger code and data space, additional instructions are required to manage paged or banked access to ROM and RAM. For example, a PIC16C77, with 4 banks of RAM, has two register page bits `RP1:RP0` which must be set or cleared prior to each register file access.[4] An optimizing compiler like PICC can remove redundant page register bit manipulation instructions in a particular function, but *each function* still needs to define the page bits so that the function's parameters and variables are accessed correctly.

Below are the memory requirements for the same program as above, but this time the Salvo variables are located in the uppermost RAM bank of each PICmicro device:

device	library used	ROM (words)	RAM (bytes)
PIC16F84	FPI400.lib	372	33
PIC16C64	FPI401.lib	374	33
PIC16C717	FPI402.lib	411	33
PIC16C65	FPI411.lib	408	33
PIC16C773	FPI412.lib	445	33
PIC16C77	FPI422.lib	479	33

Since PICC places all auto variables and parameters in RAM bank 0, bank switching is guaranteed to occur within nearly all of the Salvo functions contained in the freeware libraries. The very minor growth in code size demonstrates that the compiler will implement full register page bit control in each Salvo function *regardless of where the Salvo variables are located in memory*. That's because other functions (e.g. your own) might access variables in yet another bank, thereby changing the register page bits.

The compiler guarantees that every access to a variable will occur with the correct register page bits set. As the numbers of RAM banks and variable accesses increase, the number of instructions that will be required to set the register page bits correctly will rise accordingly.

What Can I do?

The reality is that with more RAM banks, more bank switching is required when accessing variables. Thankfully a compiler handles this for you automatically. Also, PICmicro devices with multiple RAM banks usually have more ROM than those with just one or two banks.

Ideally you should try to fit your application into the smallest possible PICmicro device, as that will usually reduce overall system cost. If you find yourself in a situation where you have exhausted the PICmicro's ROM space, yet you still have a lot of RAM available, you can configure the compiler to treat the device as if it had fewer RAM banks. Switching from 4 to 2 banks and recompiling should remove half of the bank-switching instructions in your code, which may result in the savings you're looking for.

Further Reading

For a further understanding of memory issues in the PICmicro devices, please consult:

HI-TECH Software, *PICC Lite ANSI C Compiler User's Guide*, 2000.[5]

Microchip Corporation, *PICmicro™ Mid-Range MCU Family Reference Manua*, 1997.[6]

---

1 `Salvo\Freeware\Lib\HI-TECH\FPI4xx\main.prj`, with changes to `salvocfg.h` to ensure that the program is identical for all tested devices. All Salvo variables are located in Bank 0.

2 Pre-v2.2 Salvo freeware libraries. The v2.2 and later Salvo libraries employ a different nomenclature.

3 The growth in code size of the PIC16C77 application over that of the PIC16C773 is primarily due to the former having 4 2K code pages, while the latter has only 2.

4 PICC does this with explicit `bsf` and `bcf` instructions, respectively. Each instruction occupies one instruction word in ROM and takes one instruction cycle to complete.

5 Available at HI-TECH's web site http://www.htsoft.com.

6 Available in hardcopy (if you can find it), on CD-ROM and at Microchip's web site http://www.microchip.com.