Chapter XXXV
Processor for Mobile Applications

Ben Abdallah Abderazek
University of Aizu, Japan

Arquimedes Canedo
University of Electro-Communications, Japan

Kenichi Kuroda
University of Aizu, Japan

ABSTRACT

Mobile processors are used in numerous embedded systems, including laptops, personal digital organizers, wearable computers, cellular phones, mobile Internet terminals, digital cameras, digital cam-coders, smart cards, and sensor networks nodes. Although these systems differ in terms of their communication and computation requirements, they share the common need for low power, security and small memory footprint. This chapter presents the software and hardware architecture and the design results of a low power processor architecture based on queue computation model, which offers an attractive option in the design of mobile and embedded systems.

INTRODUCTION

Embedded and mobile processor design requirements have forced computer architects to explore and discover new techniques for delivering architectures with low power consumption, low memory footprint, and fast response-time. RISC basic design has been improved in various ways to produce high performance architectures that fit in the constraints imposed by the embedded systems. A popular modification to improve RISC code density has been the utilization of a dual instruction set scheme, where an original 32-bit instruction set is available together with

a reduced instruction set of 16-bit. The available instructions in the reduced instruction set are chosen to be those instructions most frequently executed in some given applications. Having reduced instructions improves code density, and as a consequence, improves power consumption. As less bits are available in the instruction, more reduced instructions are required to execute the same task than full size instructions, leading to performance degradation.

An alternative to achieve high performance at high code density is the use of Queue-based processors (Abderaezk, Yoshinaga, & Sowa, 2006). A queue processor is a computer that uses a first-in first-out (FIFO) data structure as the intermediate storage location for computations. The FIFO data structure, called operand queue, is available through pointers located at the head and rear of the queue. Instructions implicitly reference the location where operands will be taken and the result will be stored back. Since operations have no explicit operands, the instruction set requires fewer bits than a RISC instruction set.

Queue based machines have been studied for different purposes (Preiss & Hamacher, 1985; Fernandes, Losa & Topham, 1997; Heath, Pemmara-ju & Trenk, 1996; Schmit, Levine & Ylvisaker, 2002), but none of the previous works have dealt with the benefits of the queue computing for the high performance at high code density. Furthermore, no compiler framework has been developed or studied in the literature.

In this chapter, we introduce the software and hardware development results of low power, low complexity Queue processor, named QueueCore, architecture targeted for mobile and embedded and applications. The QueueCore stores intermediate results in a circular queue-registers. Datum is inserted in the queue in produced order scheme and can be reused. This feature has a profound implication in the areas of parallel execution, programs compactness, hardware simplicity and high execution speed (Abderaezk et al., 2006).

The QueueCore (also named QC-3) instructions are 16-bit wide, simplifying fetch and decode stages and facilitating pipelining of the processor. However, the short instructions may limit the memory addressing space as only 8-bit are left for offset (6-bit) and base address (2-bit - 00:a0/d0, 01:a1/d1, 10:a2/d2, and 11:a3/d3). To cope with this shortage, QC-3 core implements *QCaEXT* technique, which uses a special "covop" instruction that extends *load* and *store* instructions offsets and also extends immediate values if necessary. The Queue processor compiler (Canedo, Abderazek & Sowa, 2006) outputs full addresses and full constants and it is the duty of the QC-3 assembler to detect and insert a "covop" instruction whenever an address or a constant exceeds the limit imposed by the instruction's field sizes. Conditional branches are handled in a particular way since the compiler does not handle target addresses, instead it generates target labels. When the assembler detects a target label, it looks if the label has been previously read and fills the instruction with the corresponding value and "covop" instruction if needed. There is a back-patch pass in the assembler to resolve all missing forward referenced instructions.

QUEUECORE COMPILER OVERVIEW

Compiling for the queue computation model differs from the conventional techniques used in compilers for register machines since queue instructions require an offset reference value rather than a location name (e.g., register number). In (Canedo, Abderazek & Sowa, 2007; Canedo, 2006), we have investigated and developed the code generation algorithm specifically for the queue computation model. In this section, we describe the design of the queue compiler infrastructure.