Chapter 10

Real-Life RISC

עד כה הצגנו DLX כמודל למעבד RISC ומעבדי P6 של Intel כדוגמה לשימוש במושגי RISC במחשבים נפוצים בשוק. בפרק הזה נציג מידע כללי על שני משפחות מעבדי RISC גם נפוצים מאוד בשוק, משפחת MIPS שמיועדת בעיקר למערכת משובצות (וגם הבסיס של מודל DLX), ומשפחת ARM שתופסת נתח שוק ענק בטלפונים וטבלטים.

Slides 2 – 6 MIPS Architecture

The main features of the MIPS family are:

RISC Instruction Set Architecture (ISA)

The basic ISA defines registers and instructions common to MIPS processors. MIPS32 defines the 32-bit RISC ISA. This is the basis for the DLX model. MIPS64 defines the 64-bit RISC ISA. It is binary compatible with MIPS32 —binary code compiled for MIPS32 will run on MIPS64.

MIPS core descriptions

Product guides for individual MIPS cores define device-dependent implementation details, such as pipeline organization, I/O organization, control registers, interfaces, programming models and so on.

Applications

MIPS core specifications are typically licensed to OEMs (original equipment manufacturers) for use in specific products. The core design is then implemented in embedded systems. MIPS-based PCs are used in China.

The **MIPS32** registers and instruction formats are shown in slide 3. These definitions are nearly identical to the DLX. In addition, MIPS32 defines logical extensions called **coprocessors**. These include a **control coprocessor** for management of virtual addressing, an interface to an **FPU**, and an open interface for **device-specific** hardware implementations. The MIPS instruction set is familiar from DLX. Slide 5 shows some convenient instructions not defined in the DLX instruction set.

The **MIPS64** register set is identical to the MIPS32 register set except that registers hold 64-bit values. All MIPS32 instructions run on MIPS64 — some instructions act on the lower 32 bits in a register and some act on the full 64-bit values. Addresses in memory are formed from a 64-bit pointer stored in a register and a 16-bit immediate value from the instruction.

MIPS registers are denoted **`$0 – $31`** (instead of **`R0 – R31`** in DLX). The MIPS assembler recognizes symbolic names for registers to aid in programming. For example, $zero refers to $0 and **`$rp`** refers to **`$31`** (as in DLX, the **JAL** instruction saves the return pointer in **`$31`**). The MIPS assembler also recognizes macros that are converted to one or more MIPS instructions. For example, the macro **`move $1,$2`** is converted to **`addi $1,$2,$0`** and the macro **`la $a0, 32-bit-address`** is converted to two machine instructions:

```
lui $1, upper 16 bits of address  (16 bits to upper 16 bits of $1)
ori $a, $1, immediate = lower 16-bits of address
```

Slides 7 – 12 ARM ISA

ARM (Advanced RISC Machine) is a processor developed by ARM Limited, which does not manufacture chips but licenses ISA designs for implementation by developers. ARM is the most widely used 32-bit RISC ISA with over 50 billion ARM processors used in phones, tablets, games, computer peripherals and other equipment. Over 98 percent of mobile phones use ARM processors.

The main features of the ARM family are:

Data Types

Byte 8 bits, Halfword 16 bits (in ARMv4 and higher) and Word (32 bits)

Standard RISC features

Load/store architecture, large uniform register file, simple addressing modes, uniform and fixed-length instruction fields and a scalar in-order pipeline.

Additional ARM architectural features

The arm defines special Shift and ALU operations, auto-increment and auto-decrement addressing modes for loops, load and store multiple instructions, conditional execution (predication) of most instructions (these cancel instructions on certain condition flags) can replace control hazards in forward jumps

The ARM is produced in 8 different versions as shown on slide 9. In addition, several classes of extension features are defined:

Jazelle — permits the ARM to run Java bytecode with no further compliations

Thumb — a reduced 16-bit version of the ISA for small devices

Debugger, ICE — testing features for embedded systems

Multiplier (64-bit result) — permits high precision integer multiplication

DSP Enhancement — defines and instructions set for common operations in digital signal processing (DSP)

Embedded systems are devices — as small as a USB flash drive or as large as a medical MRI machine — that uses one or more microprocessors but is not a general-purpose programmable computer. CPUs intended for embedded processor applications are usually chosen to be the simplest and least expensive device that meets the application requirements.

The ARM ISA defines 16 architectural registers of 32-bit width (the ARMv8 also supports 64-bit registers). These are named r0 to r15, as shown on slide 10. Registers r11 to r15 are special purpose registers, defined with similar roles to the corresponding registers in the VAX architecture —Frame Pointer, intra-procedure-call scratch register, Stack Pointer, Link Register used to return from function calls, and the Program Counter.

ARM processors operate in 7 different operating modes shown on slide 11. Besides user and supervisor modes, there are dedicated modes for DMA, IRQ, virtual memory, coprocessor emulation, and OS mode. Depending on the operating mode in which an instruction runs, it will see one of several copies of registers r8 to r14. There is a 32-bit status register visible in all operating modes. Slide 12 shows the registers visible to the instructions depending on the operating mode. For example, when switching to supervisor mode, the program counter is swapped so that no jump instruction is required to start executing OS code.

Slides 13 – 21 ARM Instructions

The ARM has several instruction sets.

The **standard instruction set** is the ARM 32-bit instructions, which are aligned on 32-bit boundaries in memory (the lowest 2 bits of the PC are always 0).

The **Thumb** instruction set contains 16-bit instructions that are 1-to-1 mapped to the 32-bit instructions. These are shortened versions with restricted options and implicit operands. For example, **add dest, src1, src2** is reduced to the form **add dest, src**. The instructions are aligned on 16-bit boundaries (the lowest bit of the PC is always 0).

The **Jazelle** instruction set executes Java bytecode directly. The ARM reads four 8-bit instructions per instruction fetch.

The ARM permits conditional execution of most instructions (predication). Slide 14 shows an example in which a conditional branch is replaced with conditional execution. This implementation of the **`if-then-else`** control block

```
r3-- ;
if ( r3 == 0 ) then r0 = r1 + r1 else r0 = r1 – r1 ;
r4 = r5 + r6 ;
```

removes the branch stalls caused by BEQ and B.

```
     SUB r3, r3, #1                          SUBS r3, r3, #1
     CMP r3, #0
     BEQ L1           ; branch on r3 == 0
     ADD r0, r1, r2   ; skipped if r3 == 0   ADDNE r0, r1, r2 ; ADD if r3 != 0
     B L2             ; jump to L2
L1:  SUB r0, r1, r2   ; skipped if r3 != 0   SUBEQ r0, r1, r2 ; SUB if r3 == 0
L2:  ADD r4, r5, r6                          ADD r4, r5, r6
```

The basic instruction set is similar to the DLX integer instructions and is shown on slide 15.

Slide 16 shows the shift and rotate instructions which permit more complex operations. For example, the expression a = b + 4 × c can be performed by left-shifting c to bits and then adding to b.

Slides 17 and 18 describes the FPU and vector extensions defined for ARM.

Slides 18 and 19 describe instructions designed for DSP (digital signal processing).

Slide 21 shows various performance improvements in the ARM with the DSP instruction set on appropriate applications.

Slide 22 Apple iPhone 5

The Apple iPhone 5 uses the ARMv7 core and the iPhone 6 uses the ARMv8. The Apple A6 is a SOC (system on a chip) designed for the iPhone, containing the ARM CPU and associated devices. Additional components of the iPhone 5 — networking and interface controllers — are shown on slide 22.

Slides 23 – 28 WinARM

WinARM is a cross-compiler for the ARM ISA that permits development of applications for ARM in C/C++ on Windows platforms. The compiler includes extensive documentation and tools such as GNU GCC compiler, GNU-Utils for compiler/linker, ARM header-files and sample applications

with source-code. The compiler permits compilation to assembly code that enables study of ARM assembly language and the ARM programming process.

Slide 24 shows a simple C program and the compiler output.

This output is analyzed in detail on slides 25 – 26. The section labeled .main (slide 25) sets up the data frame and jumps to .L2, which sets up the while() control block. The ALU operations are in section .L3 (slide 26).

Another more complex example is given on slides 27 – 28.