COMP-573A Microcomputers

Other Architectures
(ARM, IA-64, etc)

slides by Alexandre Denault

The ARM Processor

- ARM is the industry's leading provider of 32-bit embedded RISC microprocessors with almost 75% of the market .
- Here are a few examples of devices that use an ARM processor:
  - Creative - Nomad Jukebox - Digital Audio Player
  - Nike – PSA - Digital Audio Mixer
  - Sony - MZ-R90 - Minidisc Player
  - Nintendo Gameboy Advance (SP) - Handheld Games Consoles
  - Sega Dreamcast - Games Console
  - Hauppauge - WinTV - DVB-S PC TV Card
  - Sumitomo - CruiseMate - In-Car GPS

The ARM Processor (cont.)

- EMPEG - RioCar - In-Car Digital Music Player
- HP - J4169A Jet Direct 610n - Internet Print Server
- Samsung - ML6050 - Business Laser Printer
- Lexmark Z12 - Colour Inkjet Printer
- Agfa ePhoto CL18 DSC - Digital Camera
- RIM Blackberry - PDA
- 9210 & 9210i Communicator - Mobile Phone
- Ericsson/Sony Ericsson - T610 Tri-band 3GSM Mobile Phone
- HP - Jornada 520, 560, 720, 820 - Handheld PC
- HP - iPaq 5450 - Pocket PC
- Dell - AXIM X5
- Palm Tungsten T - PDA

The ARM Processor (cont.)

- Toshiba - eXX0 - Pocket PC
- Linksys - Etherfast - Firewall Router
- D-Link DWL-1000AP - 802.11B Wireless LAN Access Point
- Motorola - Jupiter - Smartcard
- Banksys - C-ZAM/SMASH - Payment Terminals
- Adaptec - 5400S - SCSI RAID for Servers
- Samsung - Compact Flash Card
- Seagate - Cheetah x15 - 15000RPM Enterprise Disk Drive
- Toshiba - 1.8" Hard Disk Drive

Features

- RISC Architecture
  - Reduced instruction set - only 25 basic instruction types.
  - Most operations are executed over registers.
- All instructions can be conditional.
- Multiple addressing modes are provided (including modes that allow direct bit shifting)
- Manual stack manipulation
  - Stack addressing must be explicitly programmed
  - Subroutines (including return) must be explicitly programmed
- Big-endian and little-endian

Memory

- 32 bit microprocessor:
  - access to memory and data manipulation made in blocks of 32 bits.
- Address range 26 bit wide :
  - allows 67,108,864 bytes (64 megabytes) of memory to be directly accessed.

Registers

- The ARM architecture has 16 registers:
  - R0-R12: 13 general purpose registers
  - R13: Stack Pointer
  - R14: Link Register
  - R15: Program Counter
    - Bit 2-25 : Next Instruction (only 24 bits because instructions are word aligned)
    - Bit 0-1 and 26-31 : Status Register

Conditional Instructions

- All ARM instructions are conditional.
- The four most significant bits of the instruction word are used to indicate one of the following 16 conditions:

EQ (EQual) 0000	HI (Higher) 1000
NE (Not Equal) 0001	LS (Lower or Same) 1001
CS (Carry Set) 0010	GE (Greater or Equal) 1010
CC (Carry Clear) 0011	LT (Less Than) 1011
MI (MInus) 0100	GT (Greater Than) 1100
PL (PLus) 0101	LE (Less than or Equal) 1101
VS (oVerflow Set) 0110	AL (Always) 1110
VC (oVerflow Clear) 0111	NV (NeVer) 1111

- If no condition is appended to the opcode menmonic, the condition AL (always) is assumed.
- Appending the condition NV (never) to any instruction effectively turns it into a ‘no operation’ instruction.

The Stack

- The ARM architecture offers extensive support for memory stack by allowing programmers to chose one of four stack format/orientation.
  - Empty or Full:
    - Empty: Stack Pointer points to the next free space on stack
    - Full: Stack Pointer points to the last item on the stack
  - Ascending or Descending:
    - Ascending: Grows from low memory to high memory
    - Descending: Grows from high memory to low memory
- I386, Sparc and PowerPC all use a *"Full, Descending"* stack format.

Instruction

- The ARM processor supports 25 different instruction. These instructions can be grouped into 4 categories:
  - Data processing instructions (18 instructions)
  - Memory instructions (4 instructions)
  - Branch instructions (2 instructions)
  - Software interrupts (1 instruction)

Data processing instructions

- These basic data manipulation instructions can be found in all architectures.
- Notice that division operations are not included in the basic instruction set.

ADC Add with Carry
ADD Add
AND Bitwise logical AND
BIC Bit Clear
CMN Compare Negated
CMP Compare
EOR Exclusive OR
MLA Multiplication with accumulate
MOV Move
MVN Move Not
MUL Multiplication
ORR Bitwise logical OR
RSB Reverse Subtract
RSC Reverse Subtract with Carry
SBC Subtract with Carry
SUB Subtract
TEQ Test Equivalence
TST Test and Mask

Memory Instruction

- As mentioned previously, ARM is a load and store architecture.
- As such, load (LDR) and store (STR) instruction are provided.
  - Load byte (LDRB) and store byte (STRB) instruction are also provided.
- ARM also provides instructions to load (LDM) and store (STM) a large number of register.
  - This is very useful when entering or exiting a subroutine.
  - With STM and LDM, the programmer must define if the registers should be stored in an ascending or descending order. The programmer must also define if the memory address supplied can be used (or it should take the next block).

Branch Instructions

- ARM provides a simple branch instruction (B). Since every instruction is conditional, there is no need for explicit conditional branching instructions.
- ARM also provides a branch with link instruction. This instruction branches and updates the link register with the return address.
- ARM provides no explicit return instruction.

Software Interrupts

- The software interrupt (SWI) instruction is the only way an ARM processor can access resources controlled by the operating system.

Shift and Rotate Operations

- Shift and rotate operations are performed by the barrel shifter, and can be appended to any of the data processing operations in the previous section.
- They are applied to operand2 of the data processing instruction before execution of its function.
  - LSL: Logical shift left
  - LSR: Logical shift right
  - ASL: Arithmetic shift left
  - ASR: Arithmetic shift right
  - ROR: Rotate right
  - RRX: Rotate right with extend

Addressing Modes

- The ARM architecture has a large variety of addressing modes:
  - Register Indirect Addressing
    - Simplest way to address memory, storing the source/destination address in a register.
      - Ex: ldr r0, r1
  - Pre-Indexed Addressing
    - Similar to register indirect addressing, the source/destination address stored in a register is offset by another value.
      - Simple register : ldr r0, [r1, r2]
      - Shifted register : ldr r0,[r1,r2,LSL#2]
      - Immediate offset : ldr r0,[r1,#-4]

Addressing Modes (cont.)

- Write back
  - When working with pre-index addressing, it is sometimes practical to save the “modified” address (like load and update on PowerPC).
  - ldr r0,[r1,r8,LSL#2]!
- Post-Indexed Addressing
  - Post-Indexed Addressing is similar to Pre-Indexed Addressing with Write back. However, the address is modified and saved only **after** the load/store operation.
- Program Counter Relative Addressing
  - The ARM architecture allows developers to address memory relative to the Program Counter (r15).

Improvement over base architecture

- Thumb Instruction Set:
  - Enables Software to be coded using short 16-bit instructions.
  - Provides typical memory savings of 35%, while retaining the benefits of a 32-bit system.
  - There is zero overhead for moving between Thumb and normal Arm state.
- DSP Instruction Set:
  - Provides up to 70% for audio DSP applications.

Improvement over base architecture (cont.)

- Jazelle Instruction Set:
  - Introduces technological infrastructure for running Java code (faster than software-based Java Virtual Machine)
  - Java execution is accelerated by 8x and power consumption is reduced by 80%.
- SIMD Extensions:
  - Increase the processing capability of ARM solutions in high-performance applications
  - Lowers power consumption required by portable, battery-powered devices

Arm Ltd

- ARM designs various 16/32-bit architecture for microprocessors and controllers.
- Those designs are then licensed to electronic companies to be used as embedded solutions.
  - For example, Intel licensed Arm technology to create the StrongArm and the Xscale processor, both used in modern PDA and network equipment.

The IA-64 Processor

- The IA-64 (a.k.a. Itanium) is Intel's solution to 64-bit computing.
- In 1994, Intel and HP announce they were working together to create this new 64-bit processor.
- Rumours say that Intel and HP secretly started the project long before 1994 and designing the IA-64 took 10 years.
- The Itanium was released to the consumer market in 2001.

IA-64 Architecture

- Based on the Explicitly Parallel Instruction Computing (EPIC) design philosophy
- The design goals were:
  - Overcome the traditional limitations of the x86
  - Focus the effort of parallel processing on the compiler.
  - Provide 64-bit addressing
  - Provide 32-bit backward compatibility (afterthought)

Memory Organization

- IA-64 defines a single, uniform, linear address space of 2^{64} bytes.
  - A single space means that both data and instructions share the same memory range.
  - Uniform means that there are no address regions with predefined functionality.
  - Linear means that the address space contains no segments; all 2^{64} bytes are consecutive.
- All code is stored in little-endian byte order in memory. Data is typically stored in little-endian byte order.
- IA-64 also provides support for big-endian operating systems.

Memory Addressing

- Memory access is provided solely through explicit register load and store instructions. (like a RISC architecture)
- Only register indirect addressing is supported.

Registers

- Over 450 registers
  - 128 general purpose registers (64-bit)
  - 128 application registers (64-bit)
  - 128 floating-point registers (82-bit)
  - 64 predicate registers (1-bit)
  - 8 branch registers (64-bit)
  - And so on …
- Each general register has an a corresponding NaT (Not a Thing) Bit. The NaT bits enable propagating validity/invalidity of a speculative load result.
- Floating-point registers use a special instance of pseudo-zero, called NaTVal. NaTVal is a floating-point register value used to propagate valid/invalid results of speculative loads of floating-point data.

Register Framing (Stack)

- The first 32 general purpose registers are globals.
  - The first general purpose register is hardwired to the value 0.
- The other 96 registers behave like a stack (like Sparc's register windows).
- You can use the ALLOC instruction to allocate a stack frame. ALLOC expects 3 parameters:
  - Number of in registers (or number of out register from previous function).
  - Number of local registers
  - Number of out registers

Register Framing (cont.)

- The Register Stack Engine (RSE) works in the background to free space in the stack registers (Sparc requires the OS to do it manually).
- In some situation, you might choose not to allocate a stack frame and only use the global registers.

Register Rotation

- In some situation (like when pipelining a loop over an array in registers), you might want to shift your register window.
- Rotate allows to shift the logical names of registers (ex: reg34 becomes reg33, reg35 becomes reg34, etc)

Instruction Set

- Instructions in IA-64 are 41 bit in length.
  - Addressing a register requires 7 bits.
- Instructions are delivered to the execution unit in bundles of 128-bit.
  - That's 3 instructions and a 5-bit template.
- Instructions are clustered in groups. These groups are collections of instructions that can be dispatched simultaneously without dependencies or interlocks.
- Groups and bundles are **not** the same things.

Instruction Set (Cont.)

- It's the compilers job to determine which instructions can be grouped together (CPU does no checking).
  - All instructions are executed in order.
- Instructions can be divided into 4 categories: integer, load/store, floating-point and branch operations.
- IA-64 opcodes are not unique. They can be reuse up to 4 times (depending on the operation unit).
- It's the job of the 5-bit template (in the bundle) to determine which unit should execute which instruction.
  - 5-bit is not enough to represent the 64 possible combinations different instructions could require
  - Intel provides 24 possible templates

Instruction Format

`[(qp)] mnemonic[.comp1][.comp2] dests = srcs`

- qp : predicate register (explained latter)
- mnemonic : name of instruction
- comp1/comp2 : completers indicating optional variations on instructions
- dests/srcs : source and destination operands, typically registers

Examples:		
Simple Instruction		add r1 = r2, r3
Predicated instruction	(p4)	add r1 = r2, r3
Instruction with immediate		add r1 = r2, r3, 1
Instruction with completer		cmp.eq p3 = r2, r4

Speculative Loads

- A program can give hints to the processor on which data it should cache.
- The processor is under no obligation to cache the data.
- However, if the opportunity presents itself (free operation unit), the processor will cache that data.

Floating Point Weirdness

- The I-64 features no FP add or FP multiplication operations. It does feature a FP multiply and add (fma).
  - To multiply, a program must call fma with an addition of zero
  - To add, a program must call fma with a multiplication of one
- The I-64 features no Integer multiplication operation. Values must be transferred to FP registers for multiplication.
- The I-64 has 82-bit FP registers. However, it records number in the IEEE 80-bit FP format.
  - The remaining 2 bits are used for extra precision for intermediate computation .

Branch Prediction

- The longer the pipeline, the more important the delay when the wrong branch is taken.
- The IA-64 architecture support two types of branch prediction:
  - Dynamic: The processor tries to determine which path will be taken.
  - Static: The programmer or the compiler informs the processor which branch will be most likely taken.

Predicated Execution

- Each instruction is tagged with a 1 bit predicate register.
- During the last cycle of the pipeline, the processor check the predicate bit.
- If the predicate bit is set to 0, the instruction is cancelled.
- Predicate bits reduce dependencies between statements, thus increasing the size of instruction groups.

Unpredicated Code	Predicated Code
cmp a,b jump EQ y=3 jump END EQ: y=4 END:	cmp.eq p1,p2=a,b p1 y=4 p2 y=3

Backward compatibility

- Itanium supports all x86 instructions in one way or another, even MMX, SSE (not SSE2), Protected, Virtual 8086, and Real mode features.
- x86 registers are mapped to gp registers.
- Switching between 32-bit and 64-bit requires a lot of overhead.
  - Before leaving 64-bits, the state of the processor must be saved.
- x86 instructions are emulated and transformed into IA-64 instructions.

References

- ARM
  http://www.arm.com
- The ARM Microprocessor
  http://www.geocities.com/wonglinhoo/Arm.htm
- IA-64 Tutorials
  http://www.cs.nmsu.edu/~rvinyard/itanium/ia64wbts/
- 64-Bit CPUs: What You Need to Know
  http://www.extremetech.com/print_article/0,3428,a%253D22477,00.asp