Overview of Computing Systems

- **RISC**
  - ARM stands for Advanced RISC Machine
    - RISC = Reduced Instruction Set Computer
  - Earliest work began in the late 60’s/early 70’s

- **RISC/ARM goals**
  - All instructions executed in single cycle
  - All instructions were the same size and had fixed format (32 bits in this case)
  - Simple to decode instructions
  - Easier to validate
  - Load/Store architecture. Data in external memory accessed with explicit instructions. All other operations (adds, subtractions, logic, etc) use only registers on the processor.

- **Feature creep**
  - Although it is a “reduced” instruction set, many instructions have been keeping up with the need for more demanding algorithms.
  - For instance, 31 instructions in Berkeley RISC-1, 46 in the second ARM processor, up to several hundred now.
Widespread Use Of ARM Today

- More than 10 billion ARM processors shipped so far.
  - Digital Cameras
  - Anti-lock disc brakes
  - Gameboy
  - Apple devices
  - Cell phones (Android)
  - So many more

Cortex-M3 Architecture

- Harvard Architecture: Separate data and instruction buses
- Cortex-M3 instruction set combines high performance typical of 32 bit processor with code density of 8 and 16 bit controllers
- Each 8-bit byte has unique address meaning the processor can read or write 8, 16, or 32 bit data
Features

• Some features visible in the previous slide
• 32 bit core
  – 32-bit address space
  – 32-bit registers
  – 32-bit shifter and ALU
  – 32-bit memory transfer

Programmer’s Model

• Registers: The most basic storage area on the chip. Can be used for data, timer, counter, addresses, etc.
  – 30 general-purpose registers (for loads and stores)
  – 6 status registers
  – A program counter
  – 37 total registers
• At one time...
  – 15 general purpose registers (r0-r14)
  – One or two status registers
  – Program counter (r15 or PC)
• All registers are 32 bits wide
• One thing many fail to understand is that these registers themselves occupy memory on the device
  – For instance, registers on ARM7TDMI are between 0x10000000-0x10000FFF
Registers

- What is a register?
  - High speed storage inside the processor
- R0-R12 are general purpose registers, contain either data or addresses
- R13 is stack pointer, points to top element of stack
- R14 is link register, used to store return location for functions (subroutines)
- R15 is the PC, points to the next instruction being fetched from memory

<table>
<thead>
<tr>
<th>Registers</th>
<th>Address Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0-R12</td>
<td>0x0000.0000-0x0003.FFFF</td>
</tr>
<tr>
<td>R13 (MSP)</td>
<td>0x2000.0000-0x2000.FFFF</td>
</tr>
<tr>
<td>R14 (LR)</td>
<td>0x4000.0000-0x41FF.FFFF</td>
</tr>
<tr>
<td>R15 (PC)</td>
<td>0xE000.0000-0xE004.0FFF</td>
</tr>
</tbody>
</table>

Special Purpose Registers

- R13, the Stack Pointer:
  - Holds the address of the stack in memory
  - Unique stack pointer in all modes (except system-shares with user)
- R14, the Link Register:
  - Subroutine return address link register
  - Unique link register in all modes (except system-shares with user)
- R15, Program counter (PC)
  - PC holds address of instruction being fetched.
  - Usually only used for long memory jumps or exception recovery
- Current Program Status Register (CPSR)
  - “State of the machine”
  - Allows programs to recover from exceptions or branch on results of an operation
Instruction Execution

- Single thread, pipelined architecture
- At any given time (clock cycle)
  - one instruction being fetched
  - another being decoded
  - another being executed

Pipelining Instructions

- Fetch: Instruction fetched from memory
- Decode: Decoding of registers used in instruction
- Execute: Registers read from Register Bank, shift and ALU operation occur, write registers back to register bank
- PC always contains the address of the instruction currently being fetched (2 instructions past what is currently being executed). During pipelining, the PC increments twice before instruction one executes
Programmer’s Model-Pipeline

Pipelining Example

- code

- pipeline

<table>
<thead>
<tr>
<th>Fetch</th>
<th>Decode</th>
<th>Execute</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDR R1, =0x01</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ADD R1,R1,#4</td>
<td>LDR R1,=0x01</td>
<td>-</td>
</tr>
<tr>
<td>SUBEQ R1,R1,#4</td>
<td>ADD R1,R1,#4</td>
<td>LDR R1,=0x01</td>
</tr>
<tr>
<td>SUB R1,R1,#7</td>
<td>SUB R1,R1,#7</td>
<td>SUBEQ R1,R1,#4</td>
</tr>
<tr>
<td>ADDMI R1,R1,#10</td>
<td>ADDS R1,R1,#2</td>
<td>ADDMI R1,R1,#10</td>
</tr>
<tr>
<td>ADDS R1,R1,#2</td>
<td>ADDS R1,R1,#2</td>
<td>ADDS R1,R1,#2</td>
</tr>
<tr>
<td>ADDEQ R1,R1,#3</td>
<td>ADDEQ R1,R1,#3</td>
<td>ADDEQ R1,R1,#3</td>
</tr>
<tr>
<td>B DN</td>
<td>B DN</td>
<td>B DN</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Number Representation

- Hex represented with prefix ‘0x’
  - Ex. \texttt{0x0FC1AB22}
- Binary represented with suffix ‘y’
  - Ex. \texttt{11010010y}
- Decimal requires no prefix or suffix.

Closed Brackets

- Closed brackets, [ ], treat what they contain as an address.
- Ex. Access data stored at memory

\begin{verbatim}
LDR r1, =dataLoc ;load the 32 bit address in r1
LDR r2, [r1] ; r1 contains an address, this loads r2 with what is stored at the address
\end{verbatim}
Program Status Register Format

- **Condition code flags**
  - N: Negative
  - Z: Zero
  - C: Carry
  - V: Overflow

- **Reserved**
- **Control bits**
  - M0-M4
  - Mode bits:
    - State bit
    - FIQ disable
    - IRQ disable

- **I/F bit:**
  - Active low interrupt set

- **T bit:**
  - Set to 1 if THUMB mode (uses 16 bit instructions)

Conditional Execution

- **Most instruction sets only allow branches to be taken conditionally.**
  - ALL ARM instructions have a condition field that determines whether or not the instruction would be executed.

  - Instructions that are not executed take up only 1 cycle - could be an advantage for long code.

  - Frequently less overhead than the usual branch or subroutine jump.

  - **Ex.** ADDEQ r0, r2,r5 ; if Z flag = 1, r0 = r2+r5
Load and Store Instruction Format

- 31st bit represents N flag, 30th bit represents Z flag, 29th bit represents C flag, 28th bit is the V flag
- I,P,U,W bits distinguish between different types of addressing modes. Won't get too specific but indicates whether immediate or non-immediate use of an operand
- L: distinguishes between a load (L = 1), or store (L = 0)
- B: distinguishes between an unsigned byte (B=1) and a word (B=0)
- Rn: specifies the base register used by addressing_mode (R0-R15)
- Rd: specifies the register whose contents are to be loaded/stored

```
       31  28  27  26  25  24  23  22  21  20  19  16  15  12  11  0
        cond  0  1  I  P  U  B  W  L  Rn  Rd  addressing_mode_specific
```

Operands

- Operands are the “variables” passed to the instruction.
- In many instructions, such as data processing instructions, instructions perform a specific operation on one or two operands.
- Operand 1 is always a register.
- Operand 2 is sent to the ALU by barrel shifter

Second Operand

- Shifted register
  - Amount to shift is contained in 5 bit instruction field
    - No overhead, shift is done free in one cycle
  - Shift is stored in bottom byte of a non-register PC
    - Takes extra cycle because ARM only has 2 read ports
Big Vs. Little Endian

- **Big Endian**: Stores most significant byte of data at lower memory address
- **Little Endian**: Stores most significant byte of data at higher memory address
- Most ARM processors are biendian, can be configured to handle both
- **Ex**: Store 16 bit number 0x03E8 at locations 0x2000.0850 and 0x2000.0851

<table>
<thead>
<tr>
<th>Big Endian</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td></td>
</tr>
<tr>
<td>0x2000.0850</td>
<td>0x03</td>
</tr>
<tr>
<td>0x2000.0851</td>
<td>0xE8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Little Endian</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td></td>
</tr>
<tr>
<td>0x2000.0850</td>
<td>0xE8</td>
</tr>
<tr>
<td>0x2000.0851</td>
<td>0x03</td>
</tr>
</tbody>
</table>