Lecture 1: Introduction
January 9th 2014
Folks

- Dr. Mark Brehob
  - Lecturer (full-time teacher) in EECS
  - My background is on the embedded systems and architecture side

- Matt Smith
  - Head lab instructor
  - Been doing 373 for more than 10 years!

- IAs:
  - Jason Shintani
  - Richard Ortman
  - Ronak Mehta
  - Ryan McMahon
What is an embedded system?
Embedded, everywhere
Embedded, Everywhere - Fitbit
What is driving the embedded everywhere explosion?
Outline

**Technology Trends**

Course Description/Overview

Tools Overview/ISA start
Bell’s Law of Computer Classes:
A new computing class roughly every decade

“Roughly every decade a new, lower priced computer class forms based on a new programming platform, network, and interface resulting in new usage and the establishment of a new industry.”

Adapted from D. Culler
Moore’s Law:
IC transistor count doubles every two years
Flash memory scaling:
Rise of density & volumes; Fall (and rise) of prices

Figure 1: 32Gb MLC NAND Flash contract price trend
Hendy’s “Law”:
Pixels per dollar doubles annually

Credit: Barry Hendy/Wikipedia
MEMS Accelerometers: Rapidly falling price and power

ADXL345
[Analog Devices, 2009]

25 µA @ 25 Hz

O(mA)

10 µA @ 10 Hz @ 6 bits
[ST Microelectronics, ann. 2009]
MEMS Accelerometer in 2012

Industry's Lowest Power MEMS Accelerometer

- 2 μA @ 100 Hz
- 300 nA wake-up mode
- Output Data Rate (Hz)
- Current Consumption (μA)
- 2.5V Supply

ADXL362

1.8 μA @ 100 Hz @ 2V supply!

ADXL362
[Analog Devices, 2012]
MEMS Gyroscope Chip

J. Seeger, X. Jiang, and B. Boser
Energy harvesting and storage: Small doesn’t mean powerless...

- Thin-film batteries
- Piezoelectric [Holst/IMEC]
- Electrostatic Energy Harvester [ICL]
- Shock Energy Harvesting
  CEDRAT Technologies
- RF [Intel]
- Clare Solar Cell
- Thermoelectric Ambient
  Energy Harvester [PNNL]
Bell’s Law, Take 2: Corollary to the Laws of Scale

Intel® 4004 processor
Introduced 1971
Initial clock speed
108 KHz
Number of transistors
2,300
Manufacturing technology
10µ

~0.1 Watt?

Photo credits: Intel, U. Michigan

Quad-Core Intel® Xeon® processor
Quad-Core Intel® Core™2 Extreme processor
Introduced 2006
Intel® Core™2 Quad processors
Introduced 2007
Initial clock speed
2.66 GHz
Number of transistors
582,000,000
Manufacturing technology
65nm

~100 Watts

UMich Phoenix Processor
Introduced 2008
Initial clock speed
106 kHz @ 0.5V Vdd
Number of transistors
92,499
Manufacturing technology
0.18 µ

15x size decrease
40x transistors
55x smaller λ

~30pW Watts?
Why study 32-bit MCUs and FPGAs?
MCU-32 and PLDs are tied in embedded market share

Source: iSupply
What differentiates these products from one another?

FPGA
=====

Microprocessor
==============
What differentiates these products from one another?
The difference is...

Peripherals

Peripherals

Peripherals

Peripherals
Why study the ARM architecture (and the Cortex-M3 in particular)?
Lots of manufacturers ship ARM products
ARM is the big player

• ARM has a huge market share
  - As of 2011 ARM has chips in about 90% of the world’s mobile handsets
  - As of 2010 ARM has chips in 95% of the smartphone market, 10% of the notebook market
    • Expected to hit 40% of the notebook market in 2015.
  - Heavy use in general embedded systems.
    • Cheap to use
      - ARM appears to get an average of 8¢ per device (averaged over cheap and expensive chips).
    • Flexible
      - Spin your own designs.
Outline

Technology Trends

Course Description/Overview

Tools Overview/ISA start
Course goals

- *Learn to implement* embedded systems including hardware/software interfacing.

- *Learn to design* embedded systems and how to think about embedded software and hardware.

- Have the opportunity to *design and build* non-trivial projects involving both hardware and software.
Prerequisites

- EECS 270: Introduction to Logic Design
  - Combinational and sequential logic design
  - Logic minimization, propagation delays, timing

- EECS 280: Programming and Intro Data Structures
  - C programming
  - Algorithms (e.g. sort) and data structures (e.g. lists)

- EECS 370: Introduction to Computer Organization
  - Basic computer architecture
  - CPU control/datapath, memory, I/O
  - Compiler, assembler
Topics

- **Memory-mapped I/O**
  - The idea of using memory addressed to talk to input and output devices.
    - Switches, LEDs, hard drives, keyboards, motors

- **Interrupts**
  - How to get the processor to become “event driven” and react to things as they happen.

- **Working with Analog inputs**
  - The real world isn’t digital!

- **Common devices and interfaces**
  - Serial buses, timers, etc.
Example: Memory-mapped I/O

- This is important.
  - It means our software can tell the hardware what to do.
    - In lab 3 you’ll design hardware on an FPGA which will control a motor.
      - But more importantly, that hardware will be designed so the software can tell the hardware exactly what to do with the motor. All by simply writing to certain memory locations!
    - In the same way, the software can read memory locations to access data from sensors etc...
Grades

<table>
<thead>
<tr>
<th>Item</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Labs (7)</td>
<td>25%</td>
</tr>
<tr>
<td>Project</td>
<td>25%</td>
</tr>
<tr>
<td>Exams</td>
<td>40% (20% midterm; 20% final)</td>
</tr>
<tr>
<td>HW/Guest talks</td>
<td>6%</td>
</tr>
<tr>
<td>Oral presentation</td>
<td>4%</td>
</tr>
</tbody>
</table>

- Project and Exams tend to be the major differentiators.
- Class median is generally a low B+. 
Time

- You’ll need to assume you are going to spend a lot of time in this class.
  - 2-3 hours/week in lecture (we cancel a few classes during project time)
  - 8-12 hours/week working in lab
    - *Expect more during project time; some labs are a bit shorter.*
  - ~20 hours (total) working on homework
  - ~20 hours (total) studying for exams.
  - ~8 hour (total) on your oral presentation

- Averages out to about 15-20 hours/week pre-project and about 20 during the project...
  - This is more than I’d like, but we’ve chosen to go with state-of-the-art tools, and those generally have a heck of a learning curve.
Labs

- 7 labs, 8 weeks, groups of 2
  1. FPGA + Hardware Tools
  2. MCU + Software Tools
  3. Memory + Memory-Mapped I/O
  4. Interrupts
  5. Timers and Counters
  6. Serial Bus Interfacing
  7. Data Converters (e.g. ADCs/DACs)

- Labs are very time consuming.
  - As noted, students estimated 8-12 hours per lab with one lab (which varied by group) taking longer.
Open-Ended Project

• Goal: learn how to build embedded systems
  - By building an embedded system
  - Work in teams of 2 to 4
  - You design your own project

• The major focus of the last third of the class.
  - Labs will be done and we will cancel some lectures and generally try to keep you focused.

• Important to start early.
  - After all the effort in the labs, it’s tempting to slack for a bit. The best projects are those that get going right away.
Homework

- 4-6 assignments
  - A few “mini” assignments
    - Mainly to get you up to speed on lab topics
  - A few “standard” assignments
    - Hit material we can’t do in lab.

- Also a small part is for showing up to guest lecturers
Looking for me?

- All office/lab hours are shared with EECS 470.
  - Monday 10-noon -- 4632 Beyster
  - Tuesday 3:30-5:00pm -- EECS 2334 (our lab)
  - Thursday 10:30-noon -- 4632 Beyster
Outline

Technology Trends

Course Description/Overview

Tools overview/ISA start
We are using Actel’s SmartFusion Evaluation Kit
A2F200M3F-FFG484ES
- 200,000 System FPGA gates, 256 KB flash memory, 64 KB SRAM, and additional distributed SRAM in the FPGA fabric and external memory controller
- Peripherals include Ethernet, DMAs, I²Cs, UARTs, timers, ADCs, DACs and additional analog resources
  - USB connection for programming and debug from Actel's design tools
  - USB to UART connection to UART_0 for HyperTerminal examples
  - 10/100 Ethernet interface with on-chip MAC and external PHY
  - Mixed-signal header for daughter card support
FPGA work
“Smart Design” configurator
Eclipse-based “Actel SoftConsole IDE”

```c
if(status & 0x01) {
    printf("Overflow latency %d\r", C-time);
}
if(status & 0x02) {
    printf("Compare latency %d\r", |1<<29| - time);
}
if(status & 0x04) {
    printf("Capture SYNC %d\r", sync_cap);
}
if(status & 0x08) {
    printf("Capture ASYNC %d\r", async_cap);
}
NVIC_ClearPendingIRQ( Fabric_IRQn );

int main() {
    NVIC_SetPriority(Fabric_IRQn, __IPRIO);
    NVIC_EnableIRQ(Fabric_IRQn);

    // Setup NVIC
    NVIC_EnableIRQ(NVIC_IRQn);
    NVIC_Init();
    NVIC_PriorityConfig(1<<31); // low time
    NVIC_PriorityConfig(1<<27); // high time
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_SetPriority(Fabric_IRQn, __IPRIO);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric нель)
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric_IRQn);
    NVIC_EnableIRQ(Fabric molec
```
Debugger is GDB-based. Includes command line. Works really quite well.
ARM ISA
Major elements of an Instruction Set Architecture
(registers, memory, word size, endianess, conditions, instructions, addressing modes)

32-bits

R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13 (SP)
R14 (LR)
R15 (PC)

mov r0, #1
ld r1, [r0,#5]
    r1=mem((r0)+5)
    bne loop
subs r2, #1

Endianness

32-bits

0xFFFFFFFF
0xE0100000
0xE0040000
0xE0000000
0xA0000000
0x60000000
0x40000000
0x20000000
0x00000000

System
Private peripheral bus - External
Private peripheral bus - Internal
External device  1.0GB
External RAM 1.0GB
Peripheral  0.5GB
SRAM  0.5GB
Code  0.5GB

Endianness
The endianess religious war: 284 years and counting!

- **Modern version**
  - Danny Cohen
  - IEEE Computer, v14, #10
  - Published in 1981
  - Satire on CS religious war

- **Historical Inspiration**
  - Jonathan Swift
  - *Gulliver's Travels*
  - Published in 1726
  - Satire on Henry-VIII’s split with the Church
  - Now a major motion picture!

- **Little-Endian**
  - LSB is at lower address

<table>
<thead>
<tr>
<th>Memory Offset</th>
<th>Value Offset (LSB) (MSB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>01 02 FF 00</td>
</tr>
<tr>
<td>0x0004</td>
<td>78 56 34 12</td>
</tr>
</tbody>
</table>

- **Big-Endian**
  - MSB is at lower address

<table>
<thead>
<tr>
<th>Memory Offset</th>
<th>Value Offset (LSB) (MSB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>01 02 00 FF</td>
</tr>
<tr>
<td>0x0004</td>
<td>12 34 56 78</td>
</tr>
</tbody>
</table>
Addressing: Big Endian vs Little Endian (370 slide)

- **Endian-ness:** ordering of bytes within a word
  - Little - increasing numeric significance with increasing memory addresses
  - Big - The opposite, most significant byte first
  - MIPS is big endian, x86 is little endian
Instruction encoding

- Instructions are encoded in machine language opcodes
- Sometimes
  - Necessary to hand generate opcodes
  - Necessary to verify assembled code is correct
- How?

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Register Value</th>
<th>Memory Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>movs r0, #10</td>
<td>001</td>
<td>00</td>
</tr>
<tr>
<td>movs r1, #0</td>
<td>001</td>
<td>00</td>
</tr>
</tbody>
</table>

Encoding T1

All versions of the Thumb ISA.

Outside IT block.

Inside IT block.
Assembly example

data:
    .byte 0x12, 20, 0x20, -1

func:
    mov r0, #0
    mov r4, #0
    movw r1, #:lower16:data
    movt r1, #:upper16:data
    top: ldrb r2, [r1],1
    add r4, r4, r2
    add r0, r0, #1
    cmp r0, #4
    bne top
Instructions used

• mov
  - Moves data from register or immediate.
  - Or also from shifted register or immediate!
    - the mov assembly instruction maps to a bunch of different encodings!
  - If immediate it might be a 16-bit or 32-bit instruction.
    - Not all values possible
    - why?

• movw
  - Actually an alias to mov.
    - “w” is “wide”
    - hints at 16-bit immediate.
From the ARMv7-M Architecture Reference Manual (posted on the website under references)

A6.7.76  MOV (register)

Move (register) copies a value from a register to the destination register. It can optionally update the condition flags based on the value.

**Encoding T1**  ARMv6-M, ARMv7-M

If \(<Rd>\) and \(<Rm>\) both from R0-R7, otherwise all versions of the Thumb ISA.

If \(<Rd>\) is the PC, must be outside or last in IT block

\[
\begin{array}{cccccccccccccc}
15 & 14 & 13 & 12 & 11 & 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & D & Rm & Rd \\
\end{array}
\]

\[
d = \text{UInt}(D;Rd); \ m = \text{UInt}(Rm); \ setflags = \text{FALSE};
\]

\[
\text{if } \ d = 15 \ \&\& \ \text{InITBlock()} \ \&\& \ \text{!LastInITBlock()} \ \text{then UNPREDICTABLE;}
\]

**Encoding T2**  All versions of the Thumb ISA.

(Formerly LSL <Rd>, <Rm>, #0)

Not permitted inside IT block

\[
\begin{array}{cccccccccccccc}
15 & 14 & 13 & 12 & 11 & 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
d = \text{UInt}(Rd); \ m = \text{UInt}(Rm); \ setFlags = \text{TRUE};
\]

\[
\text{if } \ \text{InITBlock()} \ \text{then UNPREDICTABLE;}
\]

**Encoding T3**  ARMv7-M

MOV{S}<c>.W <Rd>,<Rm>

\[
\begin{array}{cccccccccccccccc}
15 & 14 & 13 & 12 & 11 & 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & S & 1 & 1 & 1 & 1 & (0) & 0 & 0 & 0 & Rd & 0 & 0 & 0 & 0 & Rm \\
\end{array}
\]

\[
d = \text{UInt}(Rd); \ m = \text{UInt}(Rm); \ setflags = (S == '1');
\]

\[
\text{if } \ setflags \ \&\& \ (d \in \{13,15\} \ |\ | m \in \{13,15\}) \ \text{then UNPREDICTABLE;}
\]

\[
\text{if } \ !setflags \ \&\& \ (d = 15 \ |\ | m = 15 \ |\ | (d = 13 \ \&\& \ m = 13)) \ \text{then UNPREDICTABLE;}
\]

There are similar entries for move immediate, move shifted (which actually maps to different instructions) etc.
Directives

• #:lower16:data
  - What does that do?
  - Why?
A6.7.78  MOVT

Move Top writes an immediate value to the top halfword of the destination register. It does not affect the contents of the bottom halfword.

**Encoding T1**  
ARMv7-M

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>i</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>imm4</td>
<td>0</td>
<td>imm3</td>
<td>Rd</td>
</tr>
</tbody>
</table>

\[ d = \text{UInt}(Rd); \] \[ \text{imm16} = \text{imm4}:i:imm3:imm8; \]
if \[ d \text{ IN } \{13,15\} \] then UNPREDICTABLE;

**Assembler syntax**

MOVT\(<c><q>\)  \(<Rd>, \#<imm16>\)

where:

\(<c><q>\)  
See *Standard assembler syntax fields* on page A6-7.

\(<Rd>\)  
Specifies the destination register.

\(<imm16>\)  
Specifies the immediate value to be written to \(<Rd>\). It must be in the range 0-65535.

**Operation**

if ConditionPassed() then
    EncodingSpecificOperations();
    R[d]<31:16> = imm16;
    // R[d]<15:0> unchanged
Loads!

• ldrb?
• ldrsb?
So what does the program _do_?

data:
    .byte 0x12, 20, 0x20, -1

func:
    mov r0, #0
    mov r4, #0
    movw r1, #:lower16:data
    movt r1, #:upper16:data

    top:    ldrb r2, [r1],1
            add r4, r4, r2
            add r0, r0, #1
            cmp r0, #4
            bne top
Questions?
Comments?
Discussion?