Coprocessors

Outline:

- the ARM coprocessor interface
- floating-point support
- MOVE coprocessor
- CP15, CP14

☞ hands-on: system software - semaphores
Coproprocessors

Outline:

- the ARM coprocessor interface
  - floating-point support
  - MOVE coprocessor
  - CP15, CP14

hands-on: system software - semaphores
Coprocessors

- ARM supports a generic extension of its instruction set through coprocessors

  - coprocessors have:
    - private registers and data types
    - their own interpretation of instructions

  - Example coprocessors:
    - hardware floating point - the VFP10
    - on-chip cache and MMU control
    - application specific (e.g. MOVE®)
Coprocessor instructions

- follow the ARM load/store model:
  - coprocessor data processing instructions
    - operate on values in coprocessor registers
  - coprocessor data transfer instructions
    - move values between memory and coprocessor registers

- and in addition:
  - coprocessor register transfers
    - move values between ARM and coprocessor registers
Coprocessor instructions

- Coprocessor data processing instructions

```plaintext
<table>
<thead>
<tr>
<th>31</th>
<th>28 27</th>
<th>24 23</th>
<th>20 19</th>
<th>16 15</th>
<th>12 11</th>
<th>8  7</th>
<th>5  4</th>
<th>3  2</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>cond</td>
<td>1 1 1 0</td>
<td>Cop1</td>
<td>CRn</td>
<td>CRd</td>
<td>CP#</td>
<td>Cop2</td>
<td>0</td>
<td>CRm</td>
<td></td>
</tr>
</tbody>
</table>
```

CDP\(<\text{cond}>\) \(\text{Pcp, Cop1, CRd, CRn, CRm, Cop2}\)

- CP# specifies the coprocessor number:
  - it performs the operation specified by Cop1 and Cop2 on data in CRn and CRm, putting the result in CRd
  - other interpretations are possible!
Coprocessor instructions

- Coprocessor data transfer instructions

```
LDC{<cond>}{L} Pcp, CRd, <addressing mode>
STC{<cond>}{L} Pcp, CRd, <addressing mode>
```
Coproces sor instructions

- Coprocessor register transfer instructions

```markdown
MRC{<cond>} Pcp, Cop1, Rd, CRn, CRm{, Cop2}
MRC{<cond>} Pcp, Cop1, Rd, CRn, CRm{, Cop2}
```

○ move a 32-bit value between the coprocessor and ARM
  - e.g.: floating-point FIX, FLOAT and compare
  - if Rd = r15, load is to CPSR (flags only)
Coprocessor instructions

- Coprocessor register transfer instructions

<table>
<thead>
<tr>
<th>cond</th>
<th>1 1 0 0 0 1 0</th>
<th>L</th>
<th>Rn</th>
<th>Rd</th>
<th>CP#</th>
<th>Cop1</th>
<th>CRm</th>
</tr>
</thead>
</table>

load from coprocessor/store to coprocessor

- move a 64-bit value between the coprocessor and ARM

- e.g. FMDRR Dm, Rd, Rn
  - Floating point 64-bit move
    - Rd := lower half of Dm
    - Rn := upper half of Dm

  - note: registers are specified independently
Coprocessor instructions

- Later ARMs (from v5) also have:
  - CDP2
  - LDC2/STC2
  - MCR2/MCR2
  - MCRR2/MRCC2

- These are the same as above except:
  - they use the former ‘NV’ (1111) condition
  - they are always, unconditionally, executed
Coprocessor mnemonics

- Generic coprocessor mnemonics specify the fields in the instructions

  e.g. \texttt{MCR p10, 0, R1, CR2, 0 ;}

- Sometimes more informative forms exist!

  \texttt{FMSR S4, R1 ; FP Move R1 to S2}

  - \texttt{p10} is the (single precision) floating point coprocessor

- Other classes or operations have similar syntax
Coprocessor interface

- Coprocessors are attached to the ARM memory bus. They:
  - watch the instruction traffic on the bus
  - copy instructions into a pipeline
    - which mimics ARMs instruction pipeline
  - execute those instructions with the right CP#
    - though they may also decline to do so
Coprocessor interface

- Issues:
  - not all instructions entering the ARM pipeline are executed
    - those following a branch are not
  - coprocessor instructions are conditionally executed
  - the coprocessor may be absent
    - if present, it may be busy
  - the coprocessor controls the data types
Coprocessor interface

- CPI - from ARM to all coprocessors
  - coprocessor instruction
  - ARM has identified a coprocessor instruction and wishes to execute it

- CPA - from the coprocessor(s) to ARM
  - coprocessor absent
  - no coprocessor present can execute it

- CPB - from the coprocessor(s) to ARM
  - coprocessor busy
  - a coprocessor can execute it, but not yet
Coprocessor interface

- Interface timing

● shows coprocessor busy, then available

<table>
<thead>
<tr>
<th>~CPI</th>
<th>CPA</th>
<th>CPB</th>
<th>Meaning</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>–</td>
<td>–</td>
<td>Not a (taken) coprocessor operation.</td>
<td>Do nothing</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>–</td>
<td>No coprocessor recognises this operation</td>
<td>Illegal instruction trap</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Coprocessor may accept instruction in future</td>
<td>Stall pipeline</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Coprocessor committed to operation</td>
<td>Coprocessor operation</td>
</tr>
</tbody>
</table>
Coproprocessors

Outline:

- the ARM coprocessor interface
- floating-point support
- MOVE coprocessor
- CP15, CP14

hands-on: system software - semaphores
Floating-point data types

- single precision:

\[ \text{value} = (-1)^S \times 1.fraction \times 2^{(\text{exponent}-127)} \]

- double precision:

\[ \text{value} = (-1)^S \times 1.fraction \times 2^{(\text{exponent}-1023)} \]
Floating point support

- Floating-point instructions
  - map into the coprocessor instruction space
  - will use coprocessor if present …
  - … otherwise trap into software emulator

- Floating-point library
  - will not use coprocessor even if present
  - faster than software emulator
  - can be called from Thumb code
    - Thumb has no coprocessor instructions
Floating point support

- VFP10
  - vector floating-point unit
    - includes vector instructions that perform multiple operations
  - exploits ARM1020Es 64-bit cache interface
    - and later …
  - can deliver 800 MFLOPS at 400 MHz
    - one load/store and one arithmetic operation per clock cycle (in vector mode)
Floating point support

- VFP10 is coprocessor number 10
  - also CP11 if double precision implemented

- IEEE 754 subset
  - supports single (32-) and (possibly) double (64-) bit fp formats
  - most functions in hardware
    - does not support
      - remainder
      - binary $\leftrightarrow$ decimal
      - round-to-integer
VFP architecture

- 32 single precision registers
  - may overlap 16 double precision registers (bit mappings vary)
- Each ‘S’ register can hold:
  - a single precision float
  - or a 32-bit integer
- Can process short vectors as well as scalars
  - up to 8 single precision
  - up to 4 double precision
# VFP system registers

- **Floating point system ID register (FPSID)**

<table>
<thead>
<tr>
<th>Implementor</th>
<th>SW format</th>
<th>SNG</th>
<th>Architecture</th>
<th>Part number</th>
<th>Variant</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>24 23 22 21 20</td>
<td>16 15</td>
<td>8 7</td>
<td>4 3</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

- **Floating point status and control register (FPSCR)**

<table>
<thead>
<tr>
<th>NZCV</th>
<th>000 FZ</th>
<th>RMODE</th>
<th>STRIDE</th>
<th>len</th>
<th>000</th>
<th>IXE</th>
<th>UFE</th>
<th>OFE</th>
<th>DZE</th>
<th>IOE</th>
<th>000</th>
<th>IXC</th>
<th>UFC</th>
<th>OFC</th>
<th>DZC</th>
<th>IOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 30 29 28 27</td>
<td>25 24 23 22 21 20 19 18</td>
<td>16 15</td>
<td>13 12 11 10 9 8 7 5 4 3 2 1 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

  - flags, rounding, vector length, exception control …

- **Floating point exception register (FPEXC)**

<table>
<thead>
<tr>
<th>EXEN</th>
<th>000</th>
<th>000</th>
<th>000</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 30 29</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

  implementation defined
Example VFP instructions

- Load and store from/to memory
  - includes some multiple register moves
- Transfers from/to ARM registers
- Copy/negate/absolute value
  - Add/subtract/multiply/divide/square root
  - single values or short vectors
- Comparisons
- Floating point/integer conversion
Example VFP instructions

- **Floating point Add, Single precision**

  \[ \text{FADDS\{<cond>\} } \quad <\text{Sd}>, \quad <\text{Sn}>, \quad <\text{Sm}> \]

  ![Register Specifiers](image)

  - \{D, N, M\} are LSBs of register specifiers

- **Floating point Divide, Double precision**

  \[ \text{FDIVD\{<cond>\} } \quad <\text{Dd}>, \quad <\text{Dn}>, \quad <\text{Dm}> \]

  ![Register Specifiers](image)
Example VFP instructions

- Floating point Load, Single precision

  FLDS{<cond>}, <Sd>, [<Rn>, #offset*4]

- Floating point Load Multiple, Double precision

  FLDM<mode>D{<cond>}, Rn{!}, <registers>
Example VFP instructions

- Floating point Move, Single precision from Register
  
  \[
  \text{FMSR\{<\text{cond}>\}} \quad \text{<Sn>}, \quad \text{<Rd>}
  \]

- Floating point Move to Register from System Register
  
  \[
  \text{FMRX\{<\text{cond}>\}} \quad \text{<Rd>}, \quad \text{<reg>}
  \]

- \(<\text{reg}> = \{\text{FPSID, FPSCR, FPEXC}\}\)
Coprocessors

Outline:

- the ARM coprocessor interface
- floating-point support
  → MOVE coprocessor
- CP15, CP14

hands-on: system software - semaphores
MOVE® coprocessor

- A video encoding acceleration coprocessor
  - to accelerate Motion Estimation (e.g. MPEG)
  - implements an 8 x 8 byte ‘block buffer’
  - major function is SAD (Sum of Absolute Differences)
    - compare 8 x 8 pixel blocks
  - has own mnemonics (starting with ‘U’)

© 2005 PEVEIT Unit – ARM System Design

Coprocessors – v4 – 27
Coprocessors

- Outline:
  - the ARM coprocessor interface
  - floating-point support
  - MOVE coprocessor
    - CP15, CP14
  - hands-on: system software - semaphores
CP15, CP14

- These two ‘coprocessors’ are interfaces to system functions
  - outside the memory space
- **CP15** is the **system control** coprocessor
  - cache, MMU, paging, etc. control and status
  - more details in ‘Memory Hierarchy Support’ section
- **CP14** is the **debug** coprocessor
  - breakpoint, watchpoint, etc. control and status
  - more details in ‘System Development’ section
Hands-on: System Software
Semaphores

- Look further into ARM system software issues
  - Use a semaphore to perform atomic I/O

☞ Follow the ‘Hands-on’ instructions