Lecture Topics

- Today: Microarchitecture
  (H&H 7.1-7.3)
- Next: continued

Announcements

- Self-study Module #10
  (assembler processing)
- Self-study Module #11
  (ARM datapath and control)
- Project #9 (due no later than 6/26)
Final Exam

- Wednesday, 6/28 (10:20-12:20)
- 24% of course grade
- 40 multiple choice questions
- Bring MSU ID, #2 pencils
- One 8.5x11 note sheet allowed

Exam #2 Statistics

High score: 98
Low score: 62
Mean score: 84

Distribution:
- 90+ 9
- 80+ 14
- 70+ 6
- 60+ 2
ARM Microprocessor

- CP
- FP Unit
- Integer Unit
- Control Unit
- RAM

RAM: holds program -- machine instructions and data
Control Unit: manages all other units
Integer Unit: circuits for integer operations
Floating Point Unit: circuits for floating point operations
Coprocessor: specialized circuits (optional)

ARM Fetch-Execute Cycle

Fetch Phase:
- RAM[ PC ] → IR
- PC + 4 → PC

Execute Phase:
- decode IR
- take appropriate action
Fetch Phase

Fetch Phase:
RAM[ PC ] → IR
PC + 4 → PC

ARM Instructions

Simplified view -- bits 27:26 of the IR identify the category of the instruction:

- 00  data processing
- 01  data movement
- 10  branch
- 11  other
Data Processing Instructions

Sixteen different operations:

- bitwise (logical) instructions (4)
- arithmetic (add/sub) instructions (6)
- move instructions (2)
- comparison (test) instructions (4)

Three items:
operand1 op operand2 ==> result

• cond: condition under which the instruction is executed
• I: operand2 is an immediate value
• opcode: specific instruction
• S: update condition code bits (NZCV)
• Rn: first operand
• Rd: destination
• operand2: Rm (register number) or imm8 (8-bit value)
Data Processing Instructions

Execute Phase:

MUX( REG[Rm], IR(7..0) ) ==> Operand2
ALU( REG[Rn], Operand2, op ) ==> REG[Rd]

Use MUX to select between REG[Rm] and imm8; send two operands to ALU
Data Movement Instructions

Two forms of data movement:

- Load: RAM $\rightarrow$ Register
- Store: Register $\rightarrow$ RAM

The data movement instructions allow us to copy values between a memory location and a register (and vice versa)

- **cond**: condition under which the instruction is executed
- **I**: operand2 is an immediate value
- **PUBW**: specific addressing mode
- **L**: load or store (1 for load, 0 for store)
- **Rn**: first operand
- **Rd**: destination
- **operand2**: Rm (register number) or imm12 (12-bit value)
Data Movement: LOAD

Execute Phase:

\[
\text{MUX}( \text{REG}[Rm], \text{IR}(11..0) ) \Rightarrow \text{Operand2}
\]
\[
\text{ALU}( \text{REG}[Rn], \text{Operand2}, \text{add} ) \Rightarrow \text{EA}
\]
\[
\text{RAM}[\text{EA}] \Rightarrow \text{REG}[Rd]
\]

Use MUX to select second operand
Data Movement: STORE

Execute Phase:

\[ \text{MUX( REG[Rm], IR(11..0) ) } \Rightarrow \text{Operand2} \]
\[ \text{ALU( REG[Rn], Operand2, add ) } \Rightarrow \text{EA} \]
\[ \text{REG[Rd] } \Rightarrow \text{RAM[EA]} \]

Use MUX to select second operand
Branch Instructions

Control transfer instructions (CTIs):

B (15 different instructions)
BL (15 different instructions)

In reality, there is only one Branch instruction, with options for conditional execution and for linking.

- **cond**: condition under which the instruction is executed
- **L**: link (save PC in LR)
- **signed_imm_24**: 24-bit two's complement value (offset)

Execution: check NZCV bits; if condition is true, branch to a different location in the program (continue sequentially if condition is false).
Control transfer: Branch instructions

Execute Phase:

if (condition is true) then
    PC + sign_extend( IR(23..0)|00 ) ==> PC
endif

Bits 31:28 of the CPSR are the NZCV bits

<table>
<thead>
<tr>
<th>cond</th>
<th>meaning</th>
<th>NZCV state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>equal</td>
<td>Z set</td>
</tr>
<tr>
<td>0001</td>
<td>not equal</td>
<td>Z clear</td>
</tr>
<tr>
<td>0010</td>
<td>carry</td>
<td>C set</td>
</tr>
<tr>
<td>0011</td>
<td>not carry</td>
<td>C clear</td>
</tr>
<tr>
<td>0100</td>
<td>negative</td>
<td>N set</td>
</tr>
<tr>
<td>0101</td>
<td>not negative</td>
<td>N clear</td>
</tr>
<tr>
<td>0110</td>
<td>overflow</td>
<td>V set</td>
</tr>
<tr>
<td>0111</td>
<td>not overflow</td>
<td>V clear</td>
</tr>
<tr>
<td>1000</td>
<td>unsigned greater than</td>
<td>C set and Z clear</td>
</tr>
<tr>
<td>1001</td>
<td>unsigned less than or equal to</td>
<td>C clear or Z set</td>
</tr>
<tr>
<td>1010</td>
<td>signed greater than or equal to</td>
<td>N == V</td>
</tr>
<tr>
<td>1011</td>
<td>signed less than</td>
<td>N != V</td>
</tr>
<tr>
<td>1100</td>
<td>signed greater than</td>
<td>Z clear and N == V</td>
</tr>
<tr>
<td>1101</td>
<td>signed less than or equal to</td>
<td>Z set or N != V</td>
</tr>
<tr>
<td>1110</td>
<td>always</td>
<td>irrelevant</td>
</tr>
</tbody>
</table>
Execute phase for Branch instructions (revised):

if (condition is true) then
  if (L is 1) then
    PC ==> LR
  endif
  endif
  PC + sign_extend( IR(23..0)|00 ) ==> PC
endif

Single-cycle ARM

The textbook discusses a single-cycle implementation of the ARM processor (each instruction executes in one clock cycle).

To simplify the diagrams, the processor only handles a few instructions: ADD, SUB, AND, ORR, LDR, STR, B

Also, it doesn’t allow shifts and has only limited options for immediate values
In many ways, the ARM is a typical RISC microprocessor:

- Machine language instructions are all the same size and have similar formats to simplify instruction fetching and decoding
- The register file has a large number of identical registers
• It is a load/store architecture: only data movement instructions can access RAM; data manipulation instructions only operate on registers

• It is only has a few addressing modes: effective addresses for load/store instructions are always formed from a base register and an offset (in a register or an immediate value)

However, the ARM does have several very unusual features that are not found in most RISC microprocessors:

• The PC is a standard register (rather than a specialized register in the control unit)

• Most instructions can be executed conditionally

• The second operand for most instructions can be shifted
Conditional Execution

Most machine language instructions have a 4-bit "condition" field (bits 31:28)

Conditional Execution

Therefore, all instructions can be executed conditionally. For example:

\[ \text{addle \ r2, r3, r4} \]

Perform the addition, but only when the NZCV bits have the pattern for LE
Example #32

Demonstrates selective control using Branch instructions

Course website:

~cse320/Examples/example32.pdf

// if (A > 15) then
//   B = B + 1
// else
//   C = C + 1
// endif

    ldr     r0, =A
    ldr     r1, [r0]
cmp r1, #15
ble else
then:
  ldr r2, =B
  ldr r3, [r2]
  add r3, r3, #1
  str r3, [r2]
  b endif
else:
  ldr r2, =C
  ldr r3, [r2]
  add r3, r3, #1
  str r3, [r2]
endif:

Example #33

Demonstrates selective control using conditional execution

Course website:

~cse320/Examples/example33.pdf
// if (A > 15) then
//    B = B+1
// else
//    C = C+1
// endif

  ldr     r0, =A
  ldr     r1, [r0]

cmp      r1, #15
  then:
    ldrgt   r2, =B
    ldrgt   r3, [r2]
    addgt   r3, r3, #1
    strgt   r3, [r2]
  else:
    ldrle   r2, =C
    ldrle   r3, [r2]
    addle   r3, r3, #1
    strle   r3, [r2]
  endif:
Branching vs. conditional execution

A side-by-side comparison shows the difference between the two approaches

Using branch instructions, control is transferred to another area of the program

Using conditional execution, some instructions are not executed (primarily helps with pipelined implementations)
ARM Fetch-Execute Cycle (revised)

Fetch Phase:
- RAM[ PC ] → IR
- PC + 4 → PC

Execute Phase:
- decode IR
- if condition is met then
  - take appropriate action
- endif

In-line Barrel Shifter

The ARM has another unusual feature: it has a barrel shifter which allows the ALU’s second operand to be shifted.
Example

Early versions of the ARM processor did not have multiplication instructions, so the programmer had to use library functions or shifting to accomplish multiplication.

Multiplication by a power of 2 can easily be done by shifting left:

\[
8 \times N \Rightarrow \text{shift } N \text{ left by 3 positions}
\]

Example (cont)

Multiplication by other values can be done using shifting and adding:

\[
9 \times N \Rightarrow (8 + 1) \times N
\]
\[
\Rightarrow 8 \times N + N
\]

Assume \(N\) is in register \(R1\):

\[
\text{add } r2, r1, r1, \text{ lsl } #3
\]
The shift and rotate instructions are actually MOV instructions which use the barrel shifter to shift or rotate the operand.

The following both assemble into the same machine language instruction:

```
  lsl   r3, r2, #5
  mov   r3, r2, lsl #5
```

Now that we have looked at conditional execution and the in-line barrel shifter, we can review the format of machine language instructions in our subset:

- 00  data processing
- 01  data movement
- 10  branch
Summary:

Bits 31:28 ("cond") – identifies the condition under which the instruction executes

Bits 27:26 ("op") – identifies the category

Data processing and data movement:

- Bits 25:20 – the specific operation
- Bits 19:16 – Rn (the first operand)
- Bits 15:12 – Rd (the "destination" register)
- Bits 11:0 – the second operand (shifted?)