Lecture Topics

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

Announcements

- Self-study Module #11
- Project #12 (due no later than 12/7)
Final Exam

- Thursday, 12/14 (12:45-2:45 PM)
- 24% of course grade
- 40 multiple choice questions
- Bring MSU ID, #2 pencils
- One 8.5x11 note sheet (both sides)

ISA Implementations

Instruction Set Architecture: the features of a microprocessor which are visible to the assembly language programmer.

Microarchitecture: the components and interconnections which implement an ISA.
ISA Implementations

Two main strategies:

- Microprogrammed control – control signals are generated by microinstructions which reside in special ROM
- Hardwired control – control signals are generated directly by combinational and sequential logic

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 fp operations (optional)
Coprocessor: specialized circuits (optional)
ARM ISA

We'll focus on the Integer Unit (similar principles for Floating Point Unit)

Review:

- Status register (CPSR)
- 32-bit general purpose registers (r0-r15)
- 32-bit instructions

ARM Fetch-Execute Cycle

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

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

We will ignore the "other" category for now.
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

If all three items are registers:

If all three items are registers:
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:

if (condition is true) then
    MUX( REG[Rm], IR(7..0) ) ==> Operand2
    ALU( REG[Rn], Operand2, op ) ==> REG[Rd]
endif

Use MUX to select between REG[Rm] and imm8; send two operands to ALU
IR: e0462007  (sub r2, r6, r7)

IR: 11100000010001100010000000000111

cond: 1110 (always)
I: 0 (register, not imm8)
op: 0010 (SUB)
S: 0 (do not update NZCV)
rn: 0110
rm: 0111
rd: 0010

IR: e2600ff (add r0, r6, 0xff)

IR: 11100010100001100000000011111111

cond: 1110 (always)
I: 1 (imm8, not register)
op: 0100 (ADD)
S: 0 (do not update NZCV)

rn: 0110
imm8: 11111111
rd: 0000

REG[6] + 0xFF ==> REG[0]

Register numbers and imm8 field extracted simultaneously (imm8 field zero extended):

IR: 11100010100001100000000011111111
MUX: 00000000000000000000000011111111

Rn Rd Rm

18
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:

if (condition is true) then
    MUX( REG[Rm], IR(11..0) ) ==> Operand2
    ALU( REG[Rn], Operand2, add ) ==> EA
    RAM[EA] ==> REG[Rd]
endif

Use MUX to select second operand
Data Movement: STORE

Execute Phase:

if (condition is true) then
    MUX( REG[Rm], IR(11..0) ) ==> Operand2
    ALU( REG[Rn], Operand2, add ) ==> EA
    REG[Rd] ==> RAM[EA]
endif

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 twos 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
  if (L bit is on) then
    PC ==> LR
  endif
  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>
IR: aa000003 (bge endloop)
IR: 10101010000000000000000000000011
cond: 1010 (greater than or equal to)
L: 0 (do not save PC in LR)
simm24: 00000000000000000000000011

if( NZCV has pattern for GE) then
    PC + sign_extend( simm24|00) ==> PC
endif

Assume NZCV bits in CPSR: 0010
GE is true for NZCV bits

Therefore:

PC + 000000000000000000000000001000
===> PC

In hex: PC + 0000000c ==> PC
IR: aa000003  (bge endloop)

IR: 10101010000000000000000000000011

Assume NZCV bits in CPSR: 1010

GE is not true for NZCV bits

Therefore, incremented PC unchanged:

\[
PC + 00000000000000000000000000000100
\]

\[\Rightarrow PC\]

In hex:  PC + 00000040  \[\Rightarrow PC\]

---

IR: eafffff8  (b loop)

IR: 11101010111111111111111111111000

cond: 1110 (always)

L: 0 (do not save PC in LR)

simm24: 11111111111111111111111111111100

if( NZCV has pattern for Always) then

\[
PC + \text{sign\_extend}(\text{simm24} | 00) \Rightarrow PC
\]

endif
IR: eafffff8  (b loop)
IR: 1110101011111111111111111111111000
Assume NZCV bits in CPSR: 1011
Always is true for NZCV bits
Therefore:

\[
\text{PC} + 11111111111111111111111111000000 \\
\Rightarrow \text{PC}
\]

In hex: \( \text{PC} + fffffffffe0 \Rightarrow \text{PC} \)

---

IR: eb00004d  (bl display4)
IR: 1110101000000000000000000001001101
cond: 1110  (always)
L:  1 (save PC in LR)
simm24: 00000000000000001001101

if( NZCV has pattern for Always) then
\[
\text{PC} + \text{sign\_extend( simm24|00)} \Rightarrow \text{PC}
\]
endif
IR: \texttt{eb00004d} (bl display4)

IR: \texttt{111010110000000000000000001001101}

Assume NZCV bits in CPSR: 1011

Always is true for NZCV bits

Therefore:

\[
\text{PC} + \text{00000000000000000000000100110100} \\
\Rightarrow \text{PC}
\]

In hex: \( \text{PC} + 0000134 \Rightarrow \text{PC} \)

---

**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
Single-cycle Datapath and Control

MUXes to choose between alternatives:

- Second ALU operand (reg vs. imm)
- Value for Rd (ALU result vs load)

Circuits complicated by PC being R15 (general-purpose register)