Lecture Topics

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

Announcements

- Self-study Module #10
- Project #11 (due no later than 11/30)
- 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)

---

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)
The ARM supports an optional Floating Point Unit (if not present, all floating point operations must be done in software).

The FPU has its own set of registers which can be used in two ways:

- Single precision – one register
- Double precision – two registers

Floating Point Unit Registers (32 bits):
- General purpose registers (s0 – s31)
- FPSCR (floating point state register) – includes floating point condition code bits

More recent implementations support double precision computations (registers d0-d31)
Example #34

Same program as Example #15, but assume values are type “float” (rather than “int”):

\[
\text{Compute } (A*5.0 - B) \times (C+D) / 2.0 \\
\text{Store results at } X
\]

Only a handful of the instructions in the program use the Floating Point Unit.

Example #35

Program which demonstrates type conversions:

\[
\text{Compute float}(A) / \text{float}(B) \\
\text{Store results at } X
\]

A mix of integer and floating point instructions in the program.
Recent implementations of the ARM FPU support double precision operations (as well as single precision)

- d0-d15 overlap s0-s31
- d16-d31 exist separately
- circuits perform sp or dp operations

Example #36

Program which demonstrates double precision operations:

- Sum of the values 1.0, 2.0, ..., 9.0, 10.0
- Store results at X

Loop controlled by FCMPD and FMSTAT
.global main

.data
.balign 8
sum: .skip 8

.text
fmt: .asciz "\nTotal: %6.1f\n"

.balign 8
one: .double 1.0
ten: .double 10.0
zero: .double 0.0

.balign 4
main:
push {lr}

// d0: sum all values (initially 0.0)
// d1: initial value in sequence (1.0)
// d2: last value in sequence (10.0)
// d3: current value in sequence (initially 1.0)

ldr r0, =zero @ Address of 0.0
fldd d0, [r0] @ Load 0.0
ldr r1, =one @ Address of 1.0
fldd d1, [r1] @ Load 1.0
ldr r2, =ten @ Address of 10.0
fldd d2, [r2] @ Load 10.0
fldd d3, [r1] @ Copy 1.0
loop:

    fcmpd  d3, d2  ; Compare to limit
    fmstat   ; Copy NZCV from FPSCR to CPSR
    bgt     endloop

    fadd d0, d0, d3  ; Add to sum
    fadd d3, d3, d1  ; Increment current

    b       loop

endloop:

ldr    r0, =sum       ; Address of sum
fstd   d0, [r0]       ; Store sum

ldr    r0, =fmt       ; 1st arg: address
fmrrd   r1, r2, d0   ; 2nd arg: sum
bl     printf

done:

    pop    {lr}
    bx     lr

<2 lemon:~/Examples > gcc example36.s

<3 lemon:~/Examples > a.out

Total:  55.0
ARM FPU Function Conventions

Conventions about passing arguments and return values vary from one environment to another

Basic conventions:

- arguments passed in FPU registers
- return value in s0 (or d0)

Example #37

Program which demonstrates parameter passing and return value:

Function to sum values between two args

Loop controlled by FCMPS and FMSTAT
float sum( float, float );

int main()
{
  float first = 1.0;
  float last = 10.0;
  float total;

  total = sum( first, last );

  printf( "\nTotal: %6.1f\n", total );

  return 0;
}

.global sum

.text
  .balign 4
one:  .float 1.0
zero: .float 0.0

.balign 4
sum:
  push {lr}
  fcpys s2, s0 @ Copy 1st arg
  fcpys s3, s1 @ Copy 2nd arg
ldr r0, =zero       @ Address of 0.0
flds s0, [r0]       @ Load 0.0 (sum)
ldr r1, =one        @ Address of 1.0
flds s1, [r1]       @ Load 1.0 (initial)

loop:
fcmps s2, s3        @ Compare to limit
fmstat              @ Copy NZCV from
                    @ FPSCR to CPSR
bgt endloop
fadds s0, s0, s2    @ Add to sum
fadds s2, s2, s1    @ Increment current
b loop

endloop:
pop {lr}
bx lr

<3 lemon:~/Examples > gcc example37.driver.c \ example37.support.s
<4 lemon:~/Examples > a.out
Total:   55.0
Summary: ARM Floating Point

Basic FPU supports single precision floating point operations (registers s0-s310)

Advanced FPU (NEON) supports double precision operations (d0-d31)

NEON also supports 128-bit quad precision operations (q0-q15, same as d0-d31) and SIMD operations

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 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:

\[ \text{RAM} [ \text{PC} ] \rightarrow \text{IR} \]
\[ \text{PC} + 4 \rightarrow \text{PC} \]

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.
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:

- **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:

\[
\text{MUX( REG[Rm], IR(7..0) )} \Rightarrow \text{Operand2}
\]
\[
\text{ALU( REG[Rn], Operand2, op )} \Rightarrow \text{REG[Rd]}
\]

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

IR: \[ \text{1110 0000 0100 0110 0010 0000 0000 0111} \]

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

IR: e28600ff  (add r0, r6, 0xff)

IR: \[ \text{1110 0010 0100 0011 0010 0000 0000 0111 1111 1111}} \]

\[ \text{cond: 1110 (always)} \]
\[ \text{I: 1 (imm8, not register)} \]
\[ \text{op: 0100 (ADD)} \]
\[ \text{S: 0 (do not update NZCV)} \]
\[ \text{rn: 0110} \]
\[ \text{imm8: 11111111} \]
\[ \text{rd: 0000} \]
\[ \text{REG[6] + 0xFF ==> REG[0]} \]
Register numbers and imm8 field extracted simultaneously (imm8 field zero extended):

IR: 1110001010000110000000011111111
MUX: 000000000000000000000000000011111111

35