Topic 5: Arithmetic Instructions

CSE 30: Computer Organization and Systems Programming
Summer Session II

Dr. Ali Irturk
Dept. of Computer Science and Engineering
University of California, San Diego
Overview

- Variables in Assembly: Registers
- Comments in Assembly
- Addition and Subtraction in Assembly
- Memory Access in Assembly
Assembly Language

- Basic job of a CPU: execute lots of instructions
- Instructions are the primitive operations that the CPU may execute.
- Different CPUs implement different sets of instructions. The set of instructions a particular CPU implements is an Instruction Set Architecture (ISA)
  - Examples: ARM, Intel 80x86, IBM/Motorola PowerPC (Macintosh), MIPS, Intel IA64, ...
Instruction Set Architectures

- Early trend was to add more and more instructions to new CPUs to do elaborate operations
  - VAX architecture had an instruction to multiply polynomials!
- RISC philosophy (Cocke IBM, Patterson, Hennessy, 1980s) –
  Reduced Instruction Set Computing
    - Keep the instruction set small and simple, makes it easier to build fast hardware.
    - Let software (compiler) do complicated operations by composing simpler ones.
Assembly Variables: Registers

- Unlike HLL like C or Java, assembly cannot use variables
  - Why not? Keep Hardware Simple
- Assembly Operands are **registers**
  - Limited number of special locations built directly into the hardware
  - Operations can only be performed on these!
- Benefit: Since registers are directly in hardware, they are very fast (faster than 1 billionth of a second)
Assembly Variables: Registers

- Drawback: Since registers are in hardware, there are a predetermined number of them
  - Solution: ARM code must be very carefully put together to efficiently use registers

- 16 registers in ARM
  - Why 16? Smaller is faster

- Each ARM register is 32 bits wide
  - Groups of 32 bits called a **word** in ARM
  - They are referred to as: r0-r14, PC and CPSR (actually 37 registers - additional registers have similar names for other modes)
In C (and most High Level Languages) variables declared first and given a type

Example:
```
int fahr, celsius;
char a, b, c, d, e;
```

Each variable can ONLY represent a value of the type it was declared as (cannot mix and match `int` and `char` variables).

In Assembly Language, the registers have no type; operation determines how register contents are treated.
The ARM Register Set

Current Visible Registers

Abort Mode
- r0
- r1
- r2
- r3
- r4
- r5
- r6
- r7
- r8
- r9
- r10
- r11
- r12
- r13 (sp)
- r14 (lr)
- r15 (pc)
- cpsr
- spsr

Banked out Registers

- User
- FIQ
- IRQ
- SVC
- Undef

- r8
- r9
- r10
- r11
- r12
- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr

- r13 (sp)
- r14 (lr)
- spsr
Processor Modes

- The ARM has seven basic operating modes:
  - **User**: unprivileged mode under which most tasks run
  - **FIQ**: entered when a high priority (fast) interrupt is raised
  - **IRQ**: entered when a low priority (normal) interrupt is raised
  - **Supervisor**: entered on reset and when a Software Interrupt instruction is executed
  - **Abort**: used to handle memory access violations
  - **Undef**: used to handle undefined instructions
  - **System**: privileged mode using the same registers as user mode

- We will mainly use User mode. Other modes much less important for this class.

- For now, only worry about r0-r12 and treat these as registers that you can store any variable.
Comments in Assembly

- Another way to make your code more readable: comments!
- Semicolon (;) is used for ARM comments
  - anything from semicolon to end of line is a comment and will be ignored
- Note: Different from C
  - C comments have format /* comment */, so they can span many lines
Assembly Instructions

- In assembly language, each statement (called an Instruction), executes exactly one of a short list of simple commands.
- Unlike in C (and most other High Level Languages), each line of assembly code contains at most 1 instruction.
- Instructions are related to operations (\(=\), \(+\), \(-\), \(*\), \(/\)) in C or Java.
ARM Addition and Subtraction

- Syntax of Instructions:
  1  2, 3, 4
where:
  1) instruction by name
  2) operand getting result ("destination")
  3) 1st operand for operation ("source1")
  4) 2nd operand for operation ("source2")

- Syntax is rigid (for the most part):
  - 1 operator, 3 operands
  - Why? Keep Hardware simple via regularity
Addition and Subtraction of Integers

- Addition in Assembly
  - Example: `ADD r0, r1, r2` (in ARM)
    Equivalent to: `a = b + c` (in C)
    where ARM registers `r0`, `r1`, `r2` are associated with C variables `a`, `b`, `c`

- Subtraction in Assembly
  - Example: `SUB r3, r4, r5` (in ARM)
    Equivalent to: `d = e - f` (in C)
    where ARM registers `r3`, `r4`, `r5` are associated with C variables `d`, `e`, `f`
Addition and Subtraction of Integers

- How do the following C statement?
  \[ a = b + c + d - e; \]

- Break into multiple instructions
  
  ```
  ADD r0, r1, r2 ; a = b + c
  ADD r0, r0, r3 ; a = a + d
  SUB r0, $t0, r4  ; a = a - e
  ```

- Notice: A single line of C may break up into several lines of ARM.

- Notice: Everything after the semicolon on each line is ignored (comments)
Addition and Subtraction of Integers

- How do we do this?
  \[ f = (g + h) - (i + j); \]

- Use intermediate temporary register
  \[
  \begin{align*}
  \text{ADD } r0, r1, r2 &; f = g + h \\
  \text{ADD } r5, r3, r4 &; \text{temp} = i + j \\
  \text{SUB } r0, r0, r5 &; f = (g+h)-(i+j)
  \end{align*}
  \]
Immediates

- Immediates are numerical constants.
- They appear often in code, so there are ways to indicate their existence
- Add Immediate:
  \[ f = g + 10 \] (in C)
  \[ \text{ADD } r0, r1, \#10 \] (in ARM)
  where ARM registers \( r0, r1 \) are associated with C variables \( f, g \)
- Syntax similar to add instruction, except that last argument is a \#number instead of a register.
Paper and pencil example (unsigned):

\[
\begin{array}{c}
\text{Multiplicand} & 1000 & 8 \\
\text{Multiplier} & \times 1001 & 9 \\
1000 & \\
0000 & \\
0000 & \\
+1000 & \\
01001000 & \\
\end{array}
\]

\(m\ \text{bits} \times n\ \text{bits} = m + n\ \text{bit product}\)
Multiplication

- In MIPS, we multiply registers, so:
  - 32-bit value x 32-bit value = 64-bit value

- Syntax of Multiplication (signed):
  - `MUL` register1, register2, register3
Example:

- in C:  \( a = b \times c; \)
- in MIPS:
  - let b be r2; let c be r3; and let a be r0 and r1 (since it may be up to 64 bits)
  
  \[
  \text{MUL } r0, r2, r3 ; \text{ b*c only 32 bits stored}
  \]

Note: Often, we only care about the lower half of the product.

\[
\text{SMULL } r0, r1, r2, r3 ; \text{ 64 bits in r0:rl}
\]
There are 2 classes of multiply - producing 32-bit and 64-bit results

32-bit versions on an ARM7TDMI will execute in 2 - 5 cycles

- **MUL** \( r0, r1, r2 \) ; \( r0 = r1 \times r2 \)
- **MLA** \( r0, r1, r2, r3 \) ; \( r0 = (r1 \times r2) + r3 \)

64-bit multiply instructions offer both signed and unsigned versions
- For these instruction there are 2 destination registers
  - **[U|S]MULL** \( r4, r5, r2, r3 \) ; \( r5:r4 = r2 \times r3 \)
  - **[U|S]MLAL** \( r4, r5, r2, r3 \) ; \( r5:r4 = (r2 \times r3) + r5:r4 \)

Most ARM cores do not offer integer divide instructions
- Division operations will be performed by C library routines or inline shifts
Conclusion

- In MIPS Assembly Language:
  - Registers replace C variables
  - One Instruction (simple operation) per line
  - Simpler is Better
  - Smaller is Faster

- Instructions so far:
  - ADD, SUB, MUL, MULA, [U|S]MULL, [U|S]MLAL

- Registers:
  - Places for general variables: r0–r12