Overview

° C operators, operands
° Variables in Assembly: Registers
° Comments in Assembly
° Data Processing Instructions
° Addition and Subtraction in Assembly

Review C Operators/Operands (#1/2)

° Operators: +, −, *, /, % (mod);
  • \(7/4=1, 7\%4=3\)

° Operands:
  • Variables: lower, upper, fahr, celsius
  • Constants: \(0, 1000, -17, 15.4\)

° Assignment Statement:
  Variable = expression
  • Examples:
    celsius = 5*(fahr-32)/9;
    a = b+c+d-e;

C Operators/Operands (#2/2)

° 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).
Assembly Design: Key Concepts

° Keep it simple!
  • Limit what can be a variable and what can’t
  • Limit types of operations that can be done to absolute minimum
    - if an operation can be decomposed into a simpler operation, don’t include it.
    - For example, \( 7 \% 4 \) operation is complex. We break it into simpler operations in Assembly.

Assembly Variables: Registers (#1/4)

° Unlike HLL, 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

Assembly Variables: Registers (#2/4)

° 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

Assembly Variables: Registers (#3/4)

° Registers are numbered from 0 to 15
° Each register can be referred to by number or name
° Number references:
  \( r_0, r_1, r_2, \ldots r_{15} \)
° \( r_{15} = pc \) has special significant:
° \( r_{15} \) is program counter pointing to instructions being fetched from memory
Assembly Variables: Registers (#4/4)

° By convention, each register also has a name to make it easier to code

° For now:
  - r0 – r3 ➔ a1 – a4
    (correspond to C functions arguments. Used for scratch pad too!)
  - r4 – r10 ➔ v1 – v7
    (correspond to function variables)

° In general, use names to make your code more readable

Comments in Assembly

° Another way to make your code more readable: comments!

° Hash (;) is used for ARMS comments
  - anything from (;) mark to end of line is a comment and will be ignored
  - GNU ARM assembler accepts (@) instead of (;) as well

° Note: Different from C.
  - C comments have format /* comment */, so they can span many lines
  - GNU ARM assembler accepts /* comments*/ as well.

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

Data processing Instructions

° Largest category of ARM instructions, all sharing the same instruction format.

° Contains:
  - Arithmetic operations
  - Comparisons (no results saved - just set condition code flags NZCV)
  - Logical operations
  - Data movement between registers

° This is a load / store architecture
  - These instruction only work on registers, NOT memory.

° They each perform a specific operation on operands.
  4 field Format: 1 2, 3, 4
  where:
  1) operation by name
  2) operand getting result ("destination")
  3) 1st operand for operation ("source1")
  4) second operand: register or shifted register or immediate (numerical constant)
Using the Barrel Shifter: The Second Operand

**Operand 1** → **Operand 2** → **Barrel Shifter** → **ALU** → **Result**

Shift value can be either be:
- 5 bit unsigned integer
- Specified in bottom byte of another register.

**Immediate value.**
- 8 bit number
- Can be rotated right through an even number of positions.
- Assembler will calculate rotate for you from constant.

**Addition and Subtraction (#1/3)**

- **Addition in Assembly**
  - Example: `add v1, v2, v3` (in ARM)
  - Equivalent to: `a = b + c` (in C)
  - where registers `v1, v2, v3` are associated with variables `a, b, c`

- **Subtraction in Assembly**
  - Example: `sub v4, v5, v6` (in ARM)
  - Equivalent to: `d = e - f` (in C)
  - where registers `v4, v5, v6` are associated with variables `d, e, f`

**Addition and Subtraction (#2/3)**

- **How do we do this?**
  - `f = (g + h) - (i + j);`

- **Use intermediate register**
  - `add v1, v2, v3 ; f = g + h`
  - `add a1, v4, v5 ; a1 = i + j`
  - `; need to save i+j, but can’t use f, so use a1`
  - `sub v1, v1, a1 ; f=(g+h)-(i+j)`

**Addition and Subtraction (#3/3)**

- **How do the following C statement?**
  - `a = b + c + d - e;`

- **Break into multiple instructions**
  - `add v1, v2, v3 ; a = b + c`
  - `add v1, v4, v5 ; a = a + d`
  - `sub v1, v1, v5 ; a = a - e`

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

- **Notice:** Everything after the (;) mark on each line is ignored (comments)
Addition/Subtraction with Immediates (#1/2)

- Immediates are numerical constants.
- They appear often in code, so there are special instructions for them.
- **Add Immediate:**
  - add v1,v2,#10 (in ARM)
  - \( f = g + 10 \) (in C)
  - where registers \( v1, v2 \) are associated with variables \( f, g \)
- Syntax similar to `add` instruction with register, except that last argument is a number instead of a register. This number should be preceded by `(#)` symbol

Addition/Subtraction with Immediates (#2/2)

- Similarly
  - add v1,v2,#-10
  - \( f = g - 10 \) (in C)
  - where registers \( v1, v2 \) are associated with variables \( f, g \)
- OR
  - sub v1,v2,#10
  - \( f = g - 10 \) (in C)
  - where registers \( v1, v2 \) are associated with variables \( f, g \)

Data Movement Instruction

- Addition with zero is conveniently used to move content of one register to another register, so:
  - add v1,v2,#0 (in ARM)
  - \( f = g \) (in C)
  - where registers \( v1, v2 \) are associated with variables \( f, g \)
- This is so often used in code that ARM has a specific instruction for it:
  - mov v1, v2
- Another useful instruction often used to provide delay in a loop is:
  - mov v1, v1 ; this also called nop (No Operation)
- This does nothing useful

Reverse Subtraction Instruction

- Normal Subtraction:
  - Example: sub v4,v5,v6 (in ARM); \( v4 \leftarrow v5 - v6 \)
  - Equivalent to: \( d = e - f \) (in C)
  - where registers \( v4, v5, v6 \) are associated with variables \( d, e, f \)
- Reverse Subtraction:
  - Example: rsb v4,v5,v6 (in ARM); \( v4 \leftarrow v6 - v5 \)
  - Equivalent to: \( d = - (e) + f \) (in C)
  - where registers \( v4, v5, v6 \) are associated with variables \( d, e, f \)
- `rsb` is useful in many situations
COMP3221 Reading Materials (Week #3)


"And in Conclusion…"

- New Instructions:
  - add
  - sub
  - mov

- New Registers:
  - C Function Variables: v1 – v7
  - Scratch Variables: a1 – a4