Disclaimer

The slides presented here are a combination of the CS251 course notes from previous terms, the work of Xiao-Bo Li, and material from the required textbook "Computer Organization and Design, ARM Edition," by David A. Patterson and John L. Hennessy. It is being used here with explicit permission from the authors.

CS251 course policy requires students to delete all course files after the term. Therefore, please do not post these slides to any website or share them.

CS251 - Computer Organization and Design

Introduction to ARM

Instructor: Zille Huma Kamal

University of Waterloo

Spring 2023

ARM/LEG Overview

- Computers execute assembly instructions
  In binary on computer, but text form for humans
- Only simple operations
  Addition, subtraction, goto, conditional goto
- Instructions operate on two types of data
  - Registers — high speed access
  - Memory (RAM) — slow to access
- This course uses ARM
  Our ARM instructions are a subset of LEGv8, which is a subset of the actual ARMv8 instructions
- **A main goal of this course is to design hardware to execute a subset of ARM instructions**

ARM Program - Instructions and Data

- An ARM program and the data for the program are stored in Random Access Memory[1] (RAM)
- Five general formats of ARM instructions
  - Format refers to how many and what type of operands
  - Operands refer to where the data is located
  - Only two sources for data in our ARM programs: Register or Memory
- 32 general purpose registers, used like a variable in an ARM instruction
  - Each register has 64 bits, eight bytes
  - Registers are identified as X0, X1,...,X31
  - X31 (XZR) always contains 0
- RAM Memory: consists of 2^{64} bytes
  - Memory accessed with byte number from 0 to $2^{64} - 1$ as address
  - Usually grouped in 4-byte blocks called *words* or 8-byte blocks called *double-words*.
  - Most memory accesses are to addresses that are multiples of 4 for instructions or multiple of 8 for data.

[1]Often in notes, examples, assignment questions or exam, we will use the term MEM to refer to off-chip RAM

ARM Program and Program Counter (PC) for Program in Memory

- Each program instruction is one word in length
- Instruction address is multiple of four
- Often write memory program as memory byte address followed by instruction:

Memory Address	Instruction
100:	`ADD X1,X2,X3`
104:	`SUB X1,X3,X5`
108:	`ADDI X2,X12,#16`

- Often don't need address and use symbolic label of important instructions:

```
start:  ADD X1,X2,X3
        SUB X1,X3,X5
        ADDI X2,X12,#16
```

- Special register, *program counter* (PC), stores address of instruction currently executing

ARM Instructions

- R-Format:
  - 3 operands, each operand is in a general purpose register
  - Example: `ADD X1,X2,X3`
    Adds contents of X2 with the contents of X3; store result in X1
  - Example: `SUB X1,X2,X3`
    Subtracts contents of X3 from the contents of X2; store result in X1

Try this

Consider the following high-level code:

```
a = b + c - d;
```

Assume, variables `a`, `b`, `c`, and `d` are in registers X0, X1, X2 and X3, respectively.
Convert the high-level code into an ARM program using minimum ARM assembly instructions.

More ARM Instructions

- D-format:
  - 3 operands, two operands are in general purpose registers
  - One operand is in Memory. Memory address is computed as a sum of the contents of a register and a constant (an `immediate`) embedded in the instruction
- LoaD Unscaled immediate Register: `LDUR X1, [X2, #24]`
  Load data from memory address X2+24 into register X1
- STore Unscaled immediate Register (STUR): `STUR X1, [X2, #32]`
  Store data from X1 into memory at address X2+32

Try this

Convert the following HL code into an ARM program using minimum ARM assembly instructions:

```
B[5] = B[4]
```

Assume, `B[0]` is in X1.

Yet More ARM Instructions

- I-Format:
  - Two operands are in regsiters
  - One operand is a constant (an `immediate`) embedded in the instruction
  - Example: `ADDI X1, X2, #100`
    Adds *immediate* value 100 to contents of X2; store result in X1
  - Example: `SUBI X1, X2, #100`
    Subtract the *immediate* value 10 from the contents of X2; store result in X1

Think About It

- Why have both ADDI and SUBI instructions?
  Isn't `ADDI X1, X2, #-10` the same as `SUBI X1, X2, #10`?
- `immediate` cannot be negative in I-Format instructions

I-Format Instructions - Example

Try this

Convert the following HL code into an ARM program using minimum ARM assembly instructions:

```
a = a + 7
```

Assume, a is in X1.

ARM Instructions for Control Flow

- B-Format:
  - an unconditional goto statement used to change the flow in the execution of the instructions.
  - The `immediate` specifies a word address relative to program counter (PC)
  - Example: `B #28`
  - $PC = PC + 4 \times 28$

Think About It

- Why multiply by 4? What's the relationship between word address and byte address in Instruction memory?
- Can `immediate` be negative in control flow instructions? Yes.

ARM Instructions for Control Flow

- CB-Format:
  - a conditional goto statement used to change the flow in the execution of the instructions.
  - The `immediate` specifies a word address relative to program counter (PC)
  - Example: `CBZ X1,#8`
    Compare contents of register X1 to zero and if condition met set program counter to $PC + 4 \times 8$

```
if X1==0 then
      PC = PC + 4 * 8
else
      PC = PC + 4
```

  - Example: `CBNZ X1,#8`
    Compare contents in register X1 to zero and if condition met set program counter to $PC + 4 \times 8$

```
if X1 != 0 then
      PC = PC + 4 * 8
else
      PC = PC + 4
```

Control flow

- When non-goto instruction executed, PC is incremented by 4, while instruction is executing

Think About It

- Why increment PC by 4?
- This auto-increment advances the program to the next instruction

- In ARM, no conditional statements like `if`
- In ARM, no loop constructions like `for`, `while`
- Control flow handled with goto-like commands
  - unconditional goto (e.g. `B`)
  - conditional goto (e.g. `CBZ`, `CBNZ`)

Control Flow Instructions Example 1

Try this

What is the final value of PC when the following code is executed, with PC=100:

```
100:  B #3
104:  ADD X1, X2, X3
108:  SUB X1, X3, X5
112:  ADDI X2, X12, #16
```

Control Flow Instructions Example 2

Try this

Convert the following HL code into an ARM program using minimum ARM assembly instructions:

```
i = 10;
while (i > 0)
    i = i - 1;
i = -5;
```

Use register X1 for variable `i`.