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

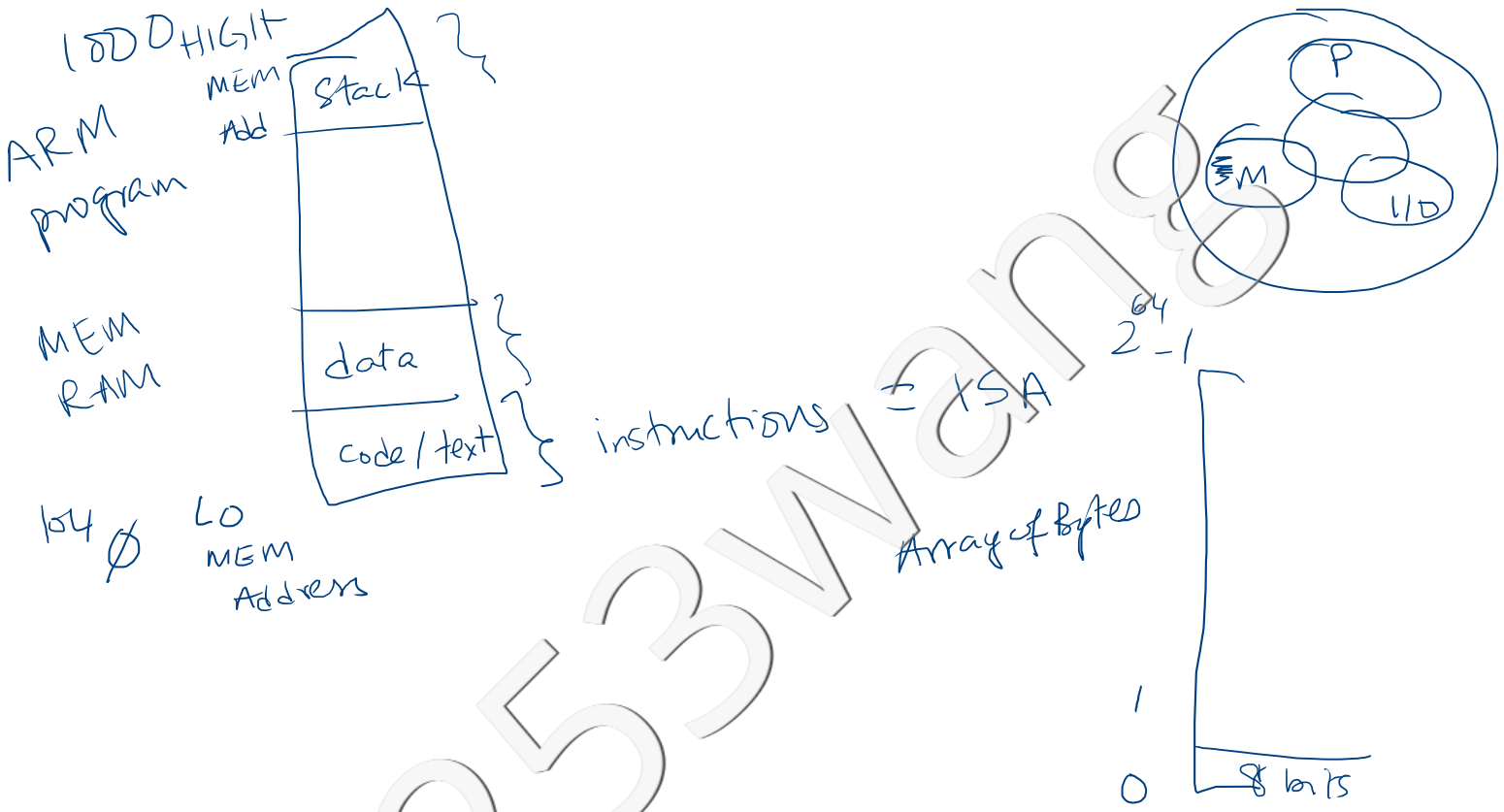
University of Waterloo

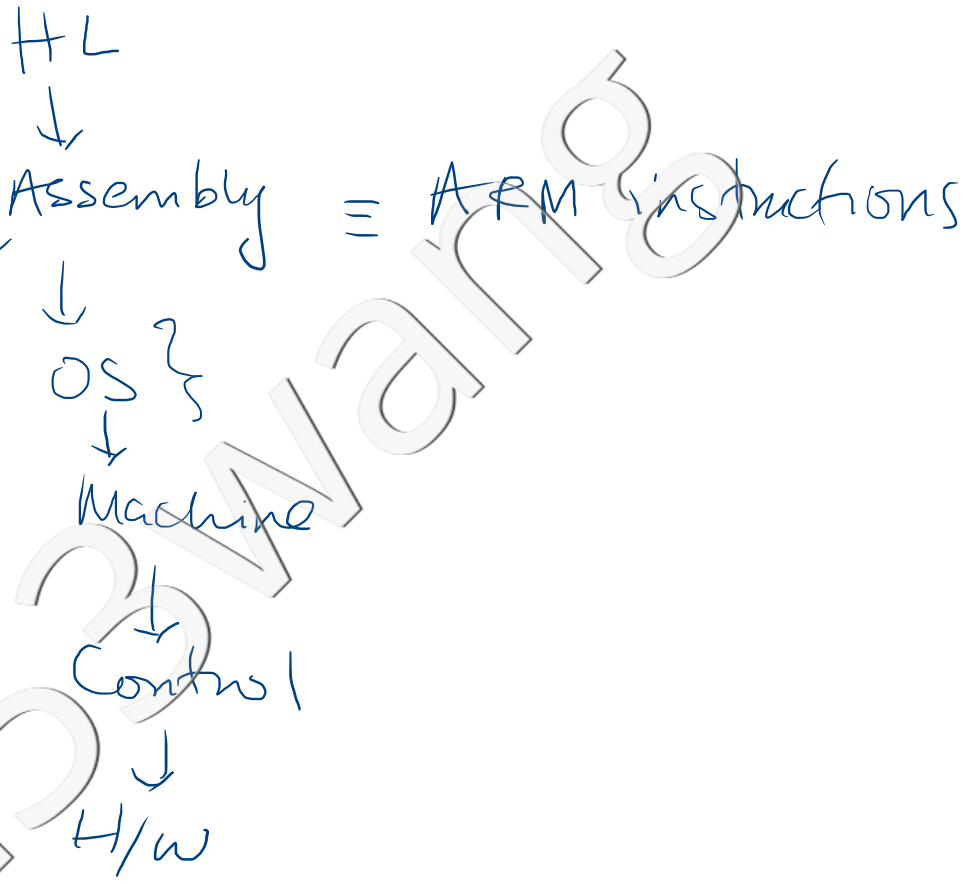
Spring 2023

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

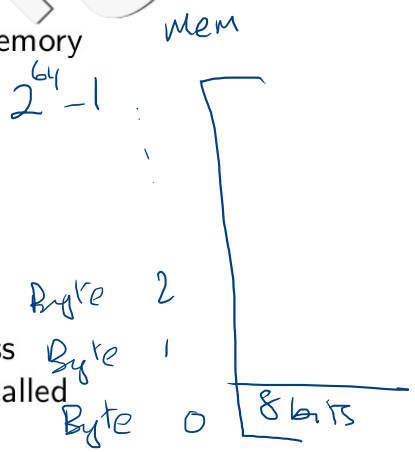
- **A main goal of this course is to design hardware to execute a subset of ARM instructions**


1000 HIGH
MEM
Add
Stack
ARM
program
MEM
RAM
data
code / text
instructions = ISA
0
LO
MEM
Address
P
SM
I/O
64
2 - 1
Array of Bytes
1
0
8 bits

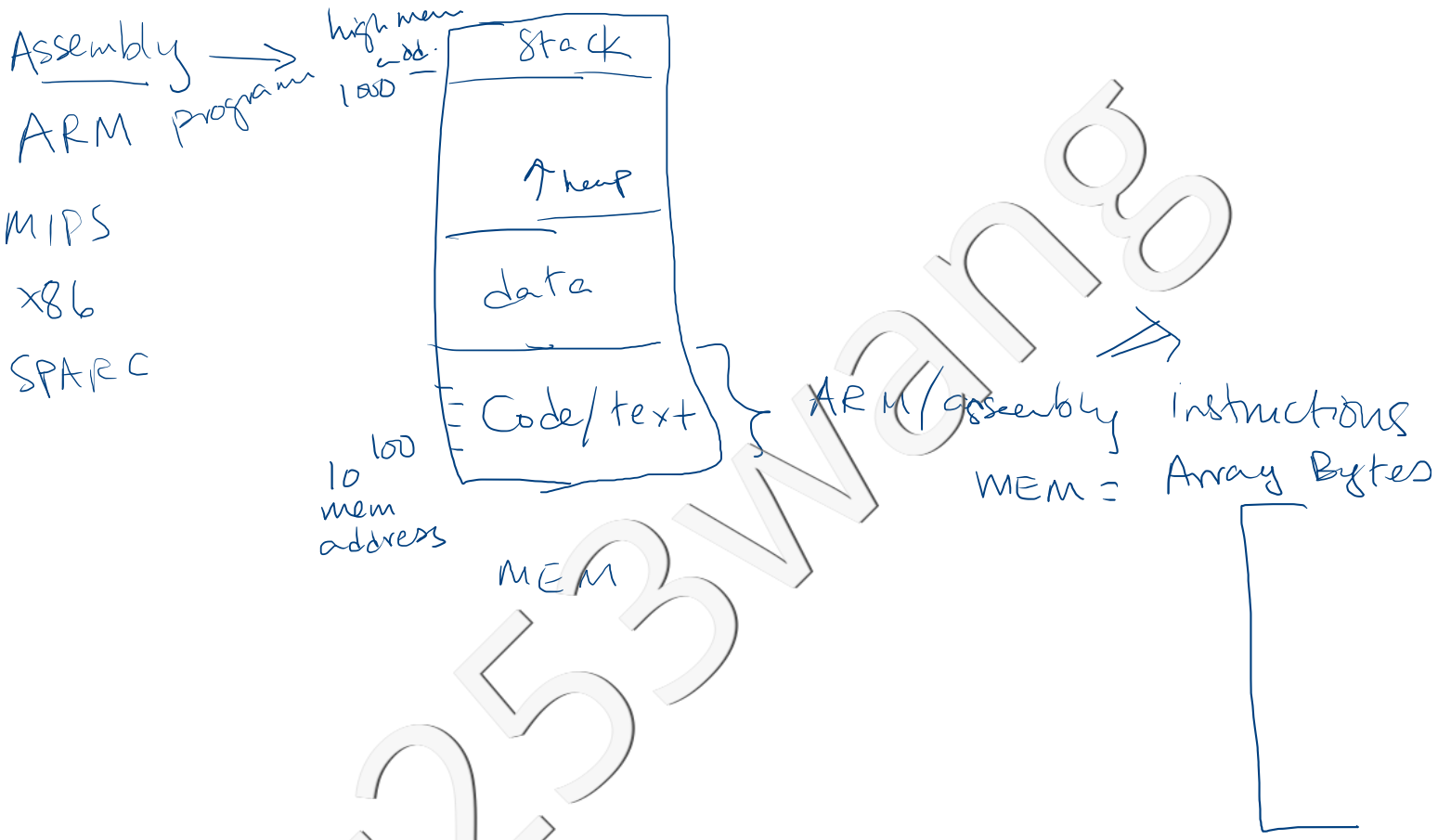


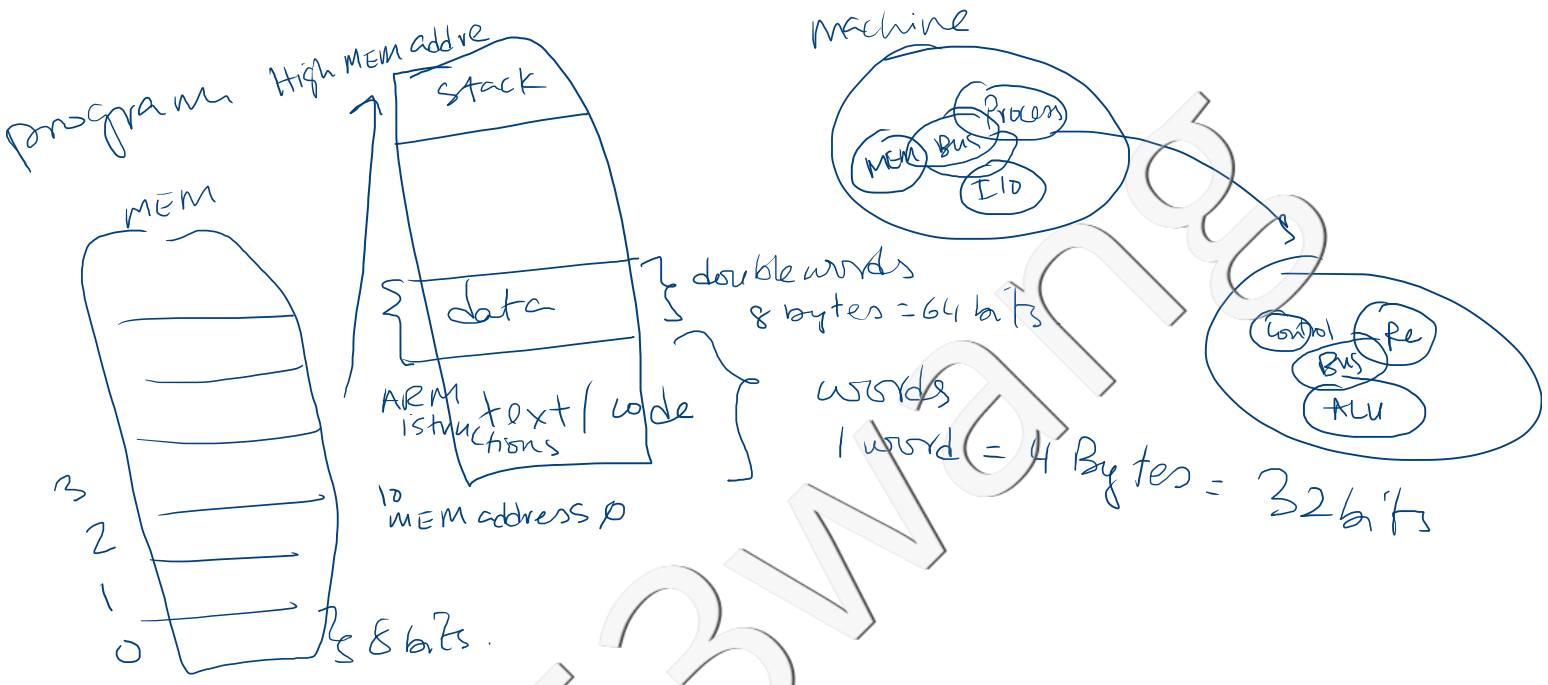
ARM Program - Instructions and Data

- 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


Assembly
ARM
MIPS
x86
SPARC
program
high mem addr.
1000
Stack
heap
data
Code/text
100
10
mem address
MEM
ARM/assembly instructions
MEM = Array Bytes


program
High MEM addre
stack
data
ARM text/code
instructions
to MEM address 0
MEM
3
2
1
0
& bits.
double words
& bytes = 64 bits
words
1 word = 4 Bytes = 32 bits
machine
Proces
MEM
Bus
I/O
control
Re
Bus
ALU

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.