University of Waterloo Territorial Acknowledgement

The University of Waterloo acknowledges that much of our work takes place on the traditional territory of the Neutral, Anishinaabeg and Haudenosaunee peoples. Our main campus is situated on the Haldimand Tract, the land granted to the Six Nations that includes six miles on each side of the Grand River. Our active work toward reconciliation takes place across our campuses through research, learning, teaching, and community building, and is co-ordinated within the Office of Indigenous Relations.

Disclaimer

The slides and lecture notes 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 CS251 and Course Outline

Instructor: Zille Huma Kamal

University of Waterloo

Fall 2024

Computer Architecture and its Organization

- Course objective: Learn organization of modern Reduced Instruction Set Computer (RISC) architecture
- Computer architecture is defined by computer organization and an Instruction Set Architecture (ISA)
- From a programmers perspective, the ISA defines the functional behaviour of the computer in the form of a set of instructions.
  - Instructions that the computer can execute
  - An instruction is a basic command e.g. `ADD r1,r2,r3`, that could represent the mathematical equation `r1=r2+r3`
- Computer organization - realization of ISA
  - Conceptual implementation of functional units,
  - Controlling interactions and exchanging information amongst the functional units.

Where does CS251 Fit in Computer Science?

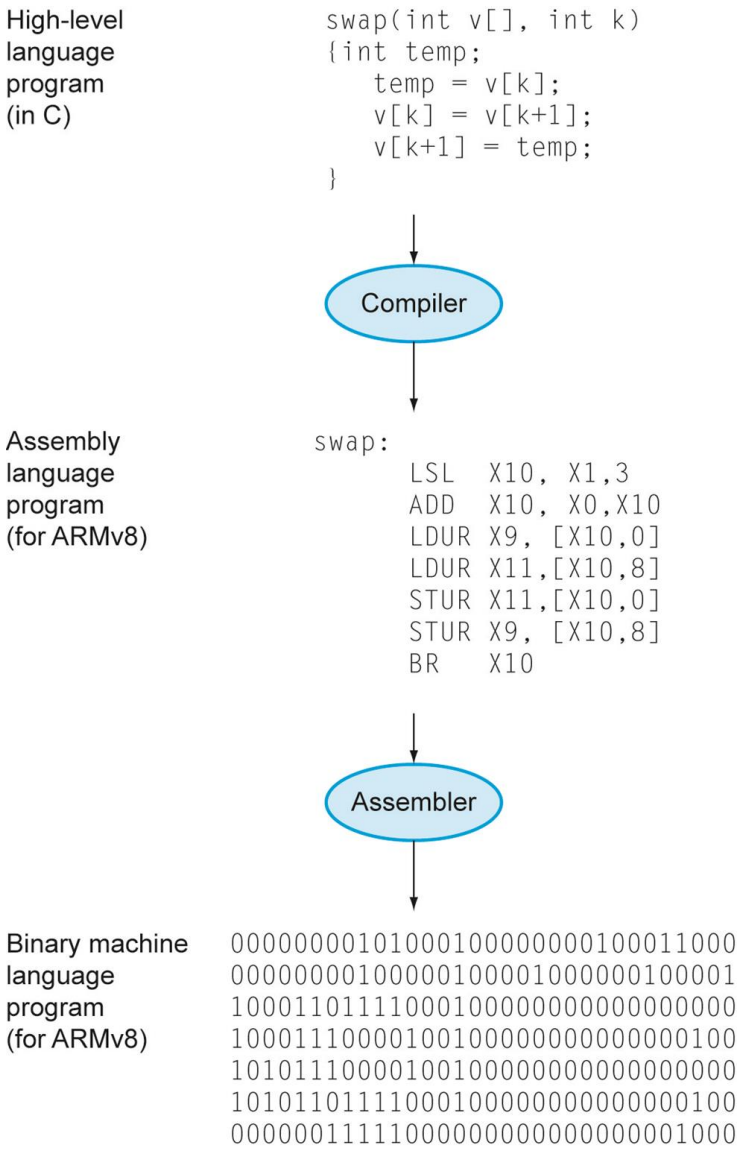


Figure 1.4 from the text

Given a basic RISC instruction, e.g. `ADD X10, X0, X10`, should be able to answer the following questions:

- **What** are the functional components needed for implementing `ADD`?
- **Where** are X0 and X10 stored?
- **How** do we get data out of X0 and X10, add it, and then store the result into X10?

With detailed discussion on the:

- transistors and gate level-logic to implement desired functionality and
- organization of the interactions between the functional components to build a processor in a computer.

Why CS251?

- Conceptual understanding of what's inside the processor of a computer
- Architecture issues influence programming
  - Example: Small code changes, big performance differences
  - Assume a 2 dimensional array, matrix, is stored in memory row-by-row
  - **Understand why** row-by-row is **faster** than column-by-column

elements accessed row-by-row

```
#include<stdio.h>
#define NR 10000
#define NC 10000

int a[NR][NC];

void main() {
  int i,j;
  for (i=0;i<NR;i++){
    for (j=0;j<NC;j++){
      a[i][j]=32767; } } }
```

elements accessed column-by-column

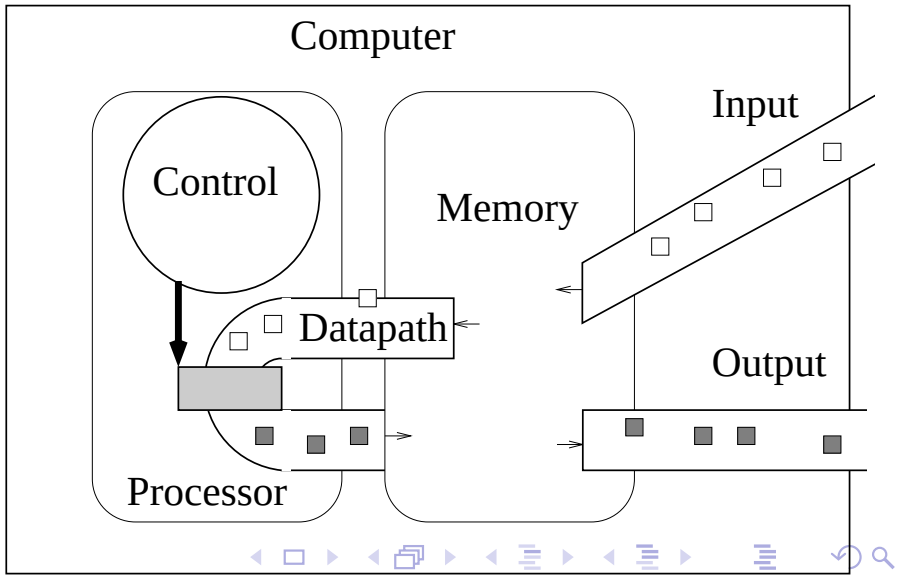
```
#include<stdio.h>
#define NR 10000
#define NC 10000

int a[NR][NC];

void main() {
  int i,j;
  for (i=0;i<NR;i++){
    for (j=0;j<NC;j++){
      a[j][i]=32767; } } }
```

Course outline

1. Introduction to course outline
   - ARM: the ISA used in this course
2. Digital logic design
   - combinational and sequential logic
3. Data representation & manipulation
   - fixed and floating points numbers
4. Single-cycle processor, designing
   - a datapath and a control unit
5. Pipelining the datapath
   - Hazards
     - ★ Structural,
     - ★ Data and
     - ★ Control
   - Exception handling
6. Memory hierarchies
   - Cache and Virtual Memory
7. Input/Output Devices
   - Hard Disk Drives (HDD) and
   - Solid State Drives (SSD)

Learning Outcomes

Design simple combinational and sequential hardware at the logic gate level in order to
Implement data-path and control unit for processors.
Describe number systems used by computer hardware, including IEEE floating point and two's complement.
Explain the limits of the binary representations, including rounding error and overflow conditions.
Analyze how machine language is executed by hardware, and
Describe a simple pipelined processor architecture for executing a RISC machine language.
Explain basic cache and virtual-memory architectures, and how they can impact performance.



CS251 Concepts in Future CS Courses

Various CS 251 topics are continued in other courses, for example,

- Memory hierarchies
  - CS 240–Data Structure and Data Management,
  - CS 350–Operating Systems,
  - CS 454–Distributed systems, and
  - CS 456–Computer Networks
- Pipelining and Exception handling
  - CS 350–Operating Systems,
  - CS 454–Distributed systems, and
  - CS 456–Computer Networks
- Uniprocessor architecture and single bus in multiprocessor systems
  - CS 343–Concurrent and Parallel Programming
- General errors caused by imperfect representation of real numbers as floating point numbers
  - CS 370–Numerical Computation

CS251 Course Material and Resources

This term, CS 251 resources will primarily be available on EdX.

- EdX support: cs-online@uwaterloo.ca
- Lecture slides **without solutions**:
  - on EdX **before** lecture, no substitute for **attending** lectures
- Lecture Scribbles - annotated lecture slides **with solutions**:
  - on EdX **after** lecture
- Lectures will be **recorded** (audio and slides only)
  - Posted by end of the day of the lecture for viewing
  - Recordings will be "raw" with automatically generated transcription
  - If you don't want to be heard in the recording: You may ask questions after lecture has ended or setup an appointment with instructor
  - No images of classroom, students or instructor in the recording.
- Required textbook:
  - "Computer Organization and Design", David Patterson and John Hennessy, ARM edition, 2017.
  - Relevant readings suggested in summary in lecture slides

Course Component, Grading Scheme: Assignments (A)

- 7 assignments to be **submitted on EdX (automarked) and Crowdmark (handmakred)**
- **No late submissions**, unless explicitly stated.
- Remark requests sent to ISA at cs251@uwaterloo.ca
  - within 1 week of release of assignment grades,
  - subject: **Remark request for A#**
  - Technical difficulties/submission issues? email your work to course account (cs251@uwaterloo.ca)

Assignment	Due Date (by 11:59pm)	Percentage
A1	Sep. 20th	2
A2	Oct. 4th	3
A3	Oct. 11th	3
A4	Nov. 1st	3
A5	Nov. 15th	3
A6	Nov. 29th	3
A7	Dec. 3rd	3

Table: Tentative Assignment Due Date and Weights.

Course Assignment Schedule and Coverage[1] at a Glance

Assignment 1 is due on September 20th and coverage is Introduction to ARM in Lectures 1 and 2.

Course Objectives

Objective	Modules	Assessment (Approx. Due Date)
Design simple combinational and sequential hardware at the logic gate level in order to **implement** simple algorithms and computations such as addition, multiplication, and datapath control unit for processors	2	Assignment 2 (Oct. 4) Assignment 3 (Oct. 11) Midterm (Oct. 22) Final (TBA)
Describe number systems used by computer hardware and **explain** the limits of the binary representations	3	Assignment 3 (Oct. 11) Midterm (Oct. 22) Final (TBA)
Midterm	1, 2, 3	Midterm (Oct. 22)
Analyze how machine language is executed by hardware	4	Midterm (Oct. 22) Assignment 4 (Nov. 1) Final (TBA)
Describe a simple pipelined processor architecture	5	Assignment 5 (Nov. 15) Final (TBA)
Explain basic cache and virtual-memory architectures, and how they can impact performance	6, 7	Assignment 6 (Nov. 29) Assignment 7 (Dec. 3) Final (TBA)
Final	All	Final (TBA)

[1] Due dates and assignment coverage are tentative

Course Component, Grading Scheme: Participation (P)

- Participation points (P) are equally weighted sum of clickers and tutorials
- Work in groups or individually
- Clickers in lectures
  - Required:
    - ★ **Free (for Fall 24) Subscription** to the iClicker app,
    - ★ **no** physical iclicker remote needed
    - ★ only register for the section you are enrolled in
  - Check your clicker marks during the term and make sure they are correct.
    - ★ No exceptions at the end of the term.
- Tutorial exercises
  - are on EdX **due at 11:59pm on Wednesdays**
- 20% of your lowest clicker and tutorial points will be discarded before computing final participation points
- Accommodates for missed classes, illness, interviews, etc.

$$\text{Participation (P)} = 2.5\% \times \text{Best 80\% of Tutorial exercise points} + 2.5\% \times \text{Best 80\% of Clicker Points}$$

Exams and Course Grades

- 1 midterm (M) exam and 1 final (F) exam
  - In-person exams
  - Midterm is on Tuesday, Oct. 22, 2024 from 4:30pm to 6:20pm.
  - Final exam information not available yet.
  - Contact ISC for **final exam relief**
  - Remark requests sent to ISA at cs251@uwaterloo.ca
    - ★ within 2 weeks of release of exam grades
    - ★ subject: **Remark request for midterm/final exam**
- Grading Scheme

$$\text{exam} = \frac{(0.25 \times M + 0.5 \times F)}{0.75}$$

$$\text{normal} = 0.2 \times A + 0.05 \times (P) + 0.25 \times M + 0.5 \times F$$

```
if (exam < 50)
    course grade = min (normal, exam)
else
    course grade = normal
```

Course Component		Points	Weight
Assignments	A1	2	20%
	A2	3	
	A3	3	
	A4	3	
	A5	3	
	A6	3	
	A7	3	
Clickers		drop 20% of your lowest scores	5%
Tutorials			
Midterm			25%
Final			50%

Online Discussion Forums

Piazza is a great way to ask questions about your assignments and course material in public.

- It enables you to learn from questions others have, and avoid asking questions that have already been asked. It also provides a forum for the instructor to make announcements and clarifications about assignments and other course related topics. As a result, students are expected to check Piazza on a regular basis.
- Link to Piazza for this term `https://piazza.com/class/m06v54bgxhy2t3`
- Follow these instructions and rules on using piazza
- All Instructor and ISA office hours for are posted in this piazza post

Copyright Issues and Intellectual Property

Copyright Issues

- Course notes contain figures from the textbook
- We have copyright permission to use them in our slides.
- Assignments and solutions contain figures from the textbook
- Student copies of course notes, solutions to assignments, etc., should not be posted or shared online

Intellectual property includes items such as

- Lecture content, spoken and written
- Questions or solution sets from assignments, clicker questions, exams
- Work protected by copyright

Must ask instructor for permission before uploading or sharing

- Online or
- With students taking the same/similar courses in subsequent terms/years

Excessive Collaboration

- In the past, as many as 10%-20% of students in the course have been caught and penalized for excessive collaboration.
- You are **encouraged** to talk to one another without writing anything down, but each student must work out his/her own assignment solution.
- Do not consult other people, other books, or Web pages.
- If you show another student your homework and we find excessive similarities between your solutions, you both will be penalized for excessive collaboration.
- Standard Penalty for first offense at Waterloo: no marks on the assignment and a deduction of 5% from the course grade, letter to associate dean.
  Additional penalties may apply depending on marking scheme.
- Standard Penalty for second offense: suspension for one term.

Excessive Collaboration

- In the past, as many as 10%-20% of students in the course have been caught and penalized for excessive collaboration.
- Previous terms: encouraged to talk/discuss, but must write up solutions on your own without checking with other students. Excessive similarities are treated as excessive collaboration.

- This term (F24): Discussions fine. Hand in own copy of assignment.
- **Caution:** Doing assignments is critical for learning the material: *"I feel lost sometimes in lectures, but the assignments help a lot."*
- **Caution:** Reading course text is critical for learning the material.

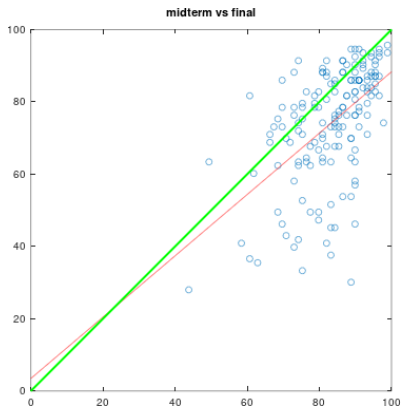
Excessive Collaboration - continued

- What's not allowed:
  - Looking for solutions on the internet
  - Using solutions from previous terms
  - Photocopying (!) another student's assignment
  - Word-for-word copying
- Standard Penalty for first offense at Waterloo: no marks on the assignment and a deduction of 5% from the course grade, letter to associate dean.
  Additional penalties may apply depending on marking scheme.
- Standard Penalty for second offense: suspension for one term.

This Course is Easier/Harder Than It Looks

Hard:

- First part - digital design - **easy!**
- Second part - pipelining, cache, VM - **HARD!**



Easy:

- Average close to 80
- **Attend** lectures, **turn off** phone and other distractions
- **do** clickers, **do** tutorials, **do** assignments

Emergency Contingency Plans

A health pandemic or other natural disasters may require shifting the in-person course to an online course.
In such scenario, the following changes may need to be made:

- Lecture notes, assigned readings and recordings of lectures will be made available online.
- Lectures **may be** held synchronously on EdX, Zoom, MS Teams, and other school provided platforms.
- Opportunity to follow course **asynchronously** using lecture recordings.
- Weekly online quizzes will be administered instead of clickers
- Exams might be online instead of in-person
- Marking scheme will be adjusted accordingly.
- Illness, quarantine

See detailed course outline on EdX for more details

Accessibility Services

Accommodations for:

- Disability
- Serious/ long-term illness or injury
- Trauma (including racial trauma)
- High-level process:
  - Apply online
  - Once eligible: *select accommodations for each course*
  - Request accommodations by assessment

Accommodations can include:

- Learning strategy resources
- Assistive technology
- Alternative testing
- Alternative format for materials
- Notetaking support
- Peer mentorship
- Accessible transportation services

Conclusion

Lecture Summary

- CS251 course outline
- Course objectives and expectations
- Course components and grading scheme

Assigned Textbook Readings

- **Skim** the following sections from Chapter 1
  - Sections 1.1, 1.2, 1.3, and 1.4

Next Steps

- **Review** the content available on EdX and note the following:
  - Course outline, road-map and calendars with tentative coverage and due date information
  - Midterm exam information
  - FYI: There was **no** in-person **tutorial** yesterday September 4th.
- **Get** the subscription for iClicker app
  - We will have a test clicker question next lecture
    - ★ only register for the section you are enrolled in

Disclaimer

The slides and lecture notes 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

Fall 2024

Instruction Set Architecture (ISA) and Assembly Instructions

- Code written in high-level languages such as Java and C++ is converted to assembly instructions that the processor can execute
- ISA defines assembly instructions (in binary format for computer, text format for humans)
- CS251 uses ARM

$$\text{ARM} \subset \text{LEGv8}^{1} \subset \text{ARMv8}$$

- Instructions encode simple operations:
  - arithmetic or logic: `ADD`, `SUB`, `AND`, `ORR`
  - data access or modification: `LDUR`, `STUR`
  - control logic: `GOTO` *label*, `UNCONDITIONAL-GOTO` *label*
- **The main goal of this course is to design hardware to execute a subset of ARM instructions**

[1] Textbook uses LEGv8

Overview of an ARM Program

- An ARM assembly program consists of instructions and data
- ARM instructions stored in:
  - Instruction Memory: consists of 2^{64} bytes (0 to $2^{64} - 1$)
  - Grouped in 4-byte blocks called a word
- Data is stored in:
  - Registers — that offer fast access time, or
  - Data Memory — that is slow to access
- Registers: 32 general purpose registers
  - Each register stores 64-bit data
  - Registers are identified as X0, X1,...,X31
  - **X31** or **XZR** always contains **0**
- Data Memory:
  - Grouped in a 8-byte blocks called double-words.

ARM Program and Program Counter (PC)

- Each ARM instruction is one word (4bytes = 32bits) in length
  - instruction addresses are always multiples of four
- Program instructions and their byte address in MEM:

Memory Address	Instruction
100:	`ADD X1,X2,X3`
104:	`SUB X1,X3,X5`

- Optionally, instead of addresses, use symbolic label of important instructions:

```
start:  ADD X1,X2,X3
        SUB X1,X3,X5
```

- Special purpose register, *program counter* (PC), stores **byte** address of the instruction that is currently executing on the processor

PC and Program Control flow

- In ARM, there are **no**
  - conditional statements like `if`
  - loop construct like `for, while`
- Program control flow is manipulated with goto-like commands
  - unconditional goto
  - conditional goto
- When non-goto commands are executed, PC is incremented by 4

Think About It

Why increment PC by 4?

ARM Instructions

- Five general formats of ARM instructions
- Format refers to how many and what type of operands
  - Registers are used like a variable in an ARM instruction, or
  - Data Memory:
    - ★ data memory accesses are always multiples of 8

Remember It

Instruction	**Format**	Example	Meaning
add	**R-format**	`ADD X1,X2,X3`	`X1 = X2 + X3`
subtract	**R-format**	`SUB X1,X2,X3`	`X1 = X2 - X3`
addi	**I-format**	`ADDI X1,X2,#C`	`X1 = X2 + C`
subi	**I-format**	`SUBI X1,X2,#C`	`X1 = X2 - C`
load word	**D-format**	`LDUR X1,[X2,#Imm]`	`X1` = Memory[`X2+Imm`]
store word	**D-format**	`STUR X1,[X2,#Imm]`	Memory[`X2+Imm`] = `X1`
branch	**B-format**	`B #Imm`	`PC = PC + 4` × `Imm`
branch on zero	**CB-format**	`CBZ X1,#Imm`	if (`X1==0`) PC = PC + 4 × Imm else PC = PC + 4
branch on non-zero	**CB-format**	`CBNZ X1,#Imm`	if (`X1!=0`) PC = PC + 4 × Imm else PC = PC + 4

- `#Imm` are signed constants, and `#C` are unsigned constants

Range of Values of #Imm and #C

Remember It

Instruction	Signed/Unsigned	Inclusive Range
ADDI/SUBI	Unsigned	$[0, 4095]$
LDUR/STUR	Signed	$[-256, 255]$
B	Signed	$[-33,554,432$ to $33,554,431]$
CBNZ/CBZ	Signed	$[-262,144$ to $262,143]$

Example 1 - Writing ARM Code

Try this

Write ARM code that accomplishes the following high-level language objective:

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

Assume, the values of the variables `a`, `b`, `c`, and `d` are in registers `X0`, `X1`, `X2` and `X3`, respectively.

Try this

Write ARM code that accomplishes the following high-level language objective:

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

Assume, the byte address of `B[0]` is in general purpose register X1.

Example 2 - Writing ARM Code

Try this

Write ARM code that accomplishes the following high-level language objective:

```
a = a + 7
b = b - 1
```

Assume, the values of the variables `a` and `b` are stored in registers `X1` and `X2`, respectively.

Think About It

- Why have both ADDI and SUBI instructions?
  Is `ADDI X1, X2, #-10` $\equiv$ `SUBI X1, X2, #10`?
- `immediate` **cannot** be negative in I-Format instructions

Example 3 - Writing ARM Code

Try this

What are the first five values of PC when the following code is executed, with PC=304:

```
PC →   304:   B #3
       308:   ADD X1, X2, X3
       312:   SUB X1, X3, X5
       316:   ADDI X2, X12,#16
       320:   B #-2
```

Try this

If PC=100, how many iterations of the loop are executed?

```
PC →   100:   ADD X1, XZR, XZR
       104:   ADDI X2, XZR, #6
       108:   ADDI X1, X1, #5
       112:   SUBI X2, X2, #1
       116:   CBNZ X2, #-2
       120:   ADD X4, X6, X8
```

Performance

- How can we compare different computer designs?
- Two important performance metrics:
  - Response time: time between start and completion of a task
  - Throughput: total number of tasks completed in a given unit of time
- Improving response time usually improves throughput
- How can we measure time?
- Nearly all computers have a clock.
- Clock cycles or ticks: discrete time when hardware events take place.
- **Clock period**: the length of a clock cycle
- **Clock rate** $= \frac{1}{\textbf{clock period}}$

Performance Metrics and Analysis

To improve performance:

- Reduce the number of clock cycles for a program
- Reduce the length of the clock cycle
- Replace a slow processor with a faster one.
  - Response time is faster, and so throughput will also increase
- Adding more processors, where each processor does **only** one task.
  - If all tasks are exactly the same, then only throughput increases.

Some factors affecting response time

- Speed of clock
- Complexity of instruction set
- Efficiency of compilers
- Mix of instructions needed to complete a task
- Some choices in designing computers:
  - simple instruction set, fast clock, one instruction executed per clock cycle
  - complex instruction set, faster clock, some instructions take multiple cycles to execute

Example: Small code changes, big performance differences

```
#include<stdio.h>
#define NR 10000
#define NC 10000

int a[NR][NC];

void main() {
  int i,j;
  for (i=0;i<NR;i++){
    for (j=0;j<NC;j++){
      a[i][j]=32767; } } }
```

- Row-by-row (a[i][j]): 1.693 sec

```
#include<stdio.h>
#define NR 10000
#define NC 10000

int a[NR][NC];

void main() {
  int i,j;
  for (i=0;i<NR;i++){
    for (j=0;j<NC;j++){
      a[j][i]=32767; } } }
```

- Down a column (a[j][i]): 27.045 sec (approx 16 times slower!)

Example Revisited: Memory access vs. Registers

```
#include<stdio.h>
#define NR 10000
#define NC 10000

int a[NR][NC];

void main() {
  register int i,j;
  for (i=0;i<NR;i++){
    for (j=0;j<NC;j++){
      ; } } }
```

- `register int i,j`: 0.044 sec

```
#include<stdio.h>
#define NR 10000
#define NC 10000

int a[NR][NC];

void main() {
  int i,j;
  for (i=0;i<NR;i++){
    for (j=0;j<NC;j++){
      ; } } }
```

- `int i,j`: 0.27 sec
  (approx 6 times slower!)

Benchmarks

- Standard set of programs (and data) chosen to measure performance
- Advantages:
  - Provides basis for meaningful comparisons
  - Design by committee may eliminate vendor bias
- Disadvantages:
  - Vendors can optimize for benchmark performance
  - Possible mismatch between benchmark and user needs
  - Still an artificial measurement

Conclusion

Lecture Summary

- ARM instruction formats and their use
- Read ARM code
- Write ARM code
- Performance metrics and measurements

Assigned Textbook Readings

- ARM Overview
- Section 1.6, just until top of page 30.
- **Skim** rest of the section

Next Steps

- **Read** assignment A1.
  - Start thinking about the questions in A1
- **Attempt** questions that are similar to the problem we solved in the lecture.
- **Ask** questions in the next tutorial or office hours.

Additional Slides

Remaining slides are additional notes for your information.

CS 251 vs CS 241

- CS 241 uses MIPS, CS 251 uses ARM.
- CS241 from HLL to Assembly
- CS251 Assembly down to hardware
- MIPS and ARM assembly similar:

MIPS	**ARM**
`lw $1, 0($2)`	`LDUR X1, [X2,#0]`
`add $1,$2,$3`	`ADD X1,X2,X3`
`addi $1,$2,22`	`ADDI X1,X2,#22`

- ARMv7 has 15 registers; ARMv8 (LEG) and MIPS have 32
  ARMv8,LEG: X31 is always 0
  MIPS: $0 is always 0
- ARM takes about 1/4 the number of transistors as MIPS
- ARMv7 has conditional forms of instructions
  ARM compiler takes advantage of this; gcc does not
  ARMv8 has fewer (than ARMv7) fancy features; we use none of them

ARM Register Format (R-Format) instructions

- Perform arithmetic or logic operations on data (operands) in registers
- Examples:
  - `ADD X1,X2,X3` $\equiv X1 = X2 + X3$
    Adds contents of X2 with the contents of X3; store result in X1.
  - `SUB X1,X2,X3` $\equiv X1 = X2 - X3$
    Subtracts contents of X3 from the contents of X2; store result in X1

ARM Data Format (D-Format) Instructions

- Move data between memory and registers
- Examples:
  - `LDUR X1, [X2, #24]` $\equiv$ X1 $\leftarrow$ MEM[X2 + 24]
    Load[2] data stored in memory byte address X2+24 into register X1.
  - `STUR X1, [X2, #32]` $\equiv$ X1 $\rightarrow$ MEM[X2 + 24]
    Store[3] contents of register X1 into memory at byte address X2+32
- Ensure data memory byte address is a multiple of 8

[2] LoaD Unscaled Register (LDUR)
[3] STore Unscaled Register (STUR)

ARM Immediate Format (I-Format) Instructions

- One operand is in a register, one operand is a constant (an `immediate`) embedded in the instruction
- Examples:
  - `ADDI X1, X2, #100` $\equiv X1 = X2 + 100$
    Adds constant 100 to contents of X2; store result in X1
  - `SUBI X1, X2, #10` $\equiv X1 = X2 - 10$
    Subtract constant 10 from the contents of X2; store result in X1

Think About It

- Why have both ADDI and SUBI instructions?
  Is `ADDI X1, X2, #-10` $\equiv$ `SUBI X1, X2, #10`?
- `immediate` **cannot** be negative in I-Format instructions

ARM Branch Format (B-Format) Instructions for Control Flow

- an **unconditional** goto statement is used to change the control flow in the execution of the instructions.
- Example: `B #28` $\equiv$ PC = PC + 4 $\times$ 28
- The `immediate` specifies a word offset relative to **current** PC

Think About It

- Why multiply by 4?
  That is the relationship between a word and a byte
- Can `immediate` be negative in control flow instructions?

ARM Conditional Branch Format (CB-Format) Instructions

- a **conditional** goto statement is used to change the flow in the execution of the instructions.
- Compare the contents of a general purpose register to **ZERO**
- Examples:

`CBZ X1,#8` is equivalent to

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

`CBNZ X1,#8` is equivalent to

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

- The `immediate` specifies a word offset relative to **current** PC