

ENGN2219/COMP6719

Computer Systems & Organization

Convener: Shoaib Akram

shoaib.akram@anu.edu.au



Australian
National
University

Plan: Week 5

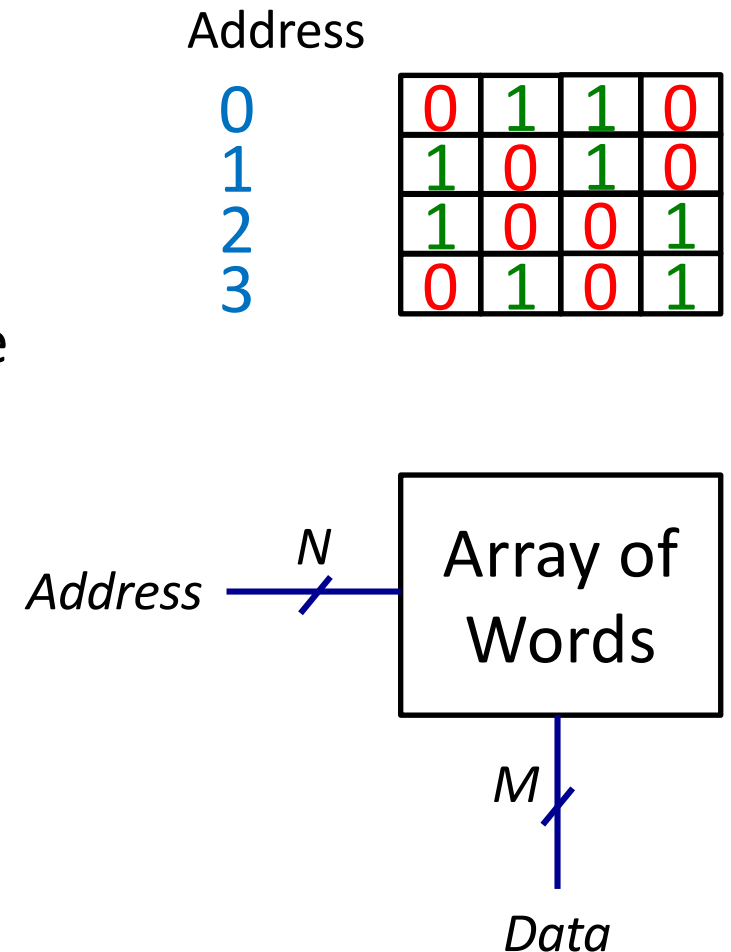
Last week: Finite State Machines, Timing, Pipelining

This Week: Finish looking at memories

This Week: Instruction Set Architecture (ISA)

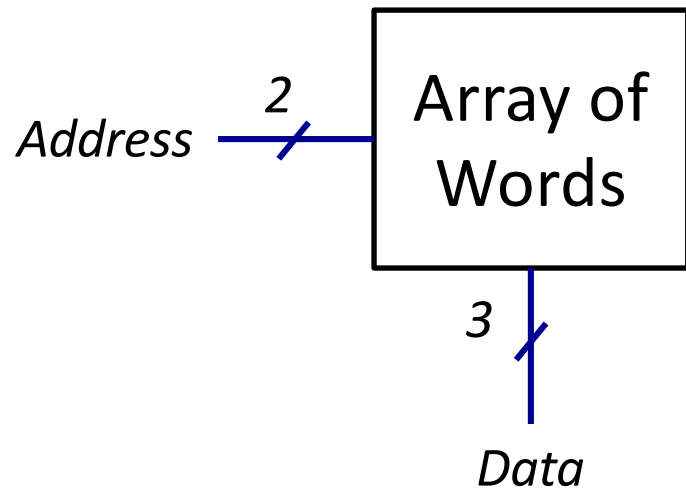
Memory

- A two-dimensional array of memory cells
 - Each cell contains one bit
 - A group of cells make up a row (word)
 - There are many rows, each with a unique address
- Memory dimensions
 - Each row is M-bits wide (Data)
 - The address is N bits: 2^N rows
 - The array contains 2^N M-bit words



Example

- 2-bit address and 3-bit words



Address	Data			height
11	0	1	1	
10	1	1	0	
01	1	1	1	
00	1	0	0	
width				

Address and Data

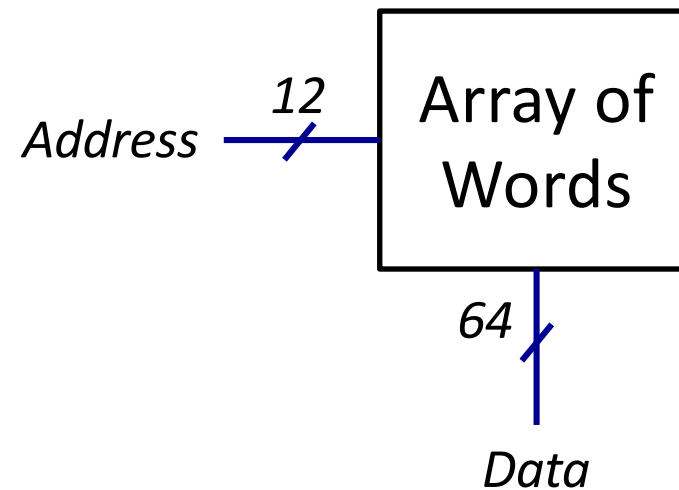
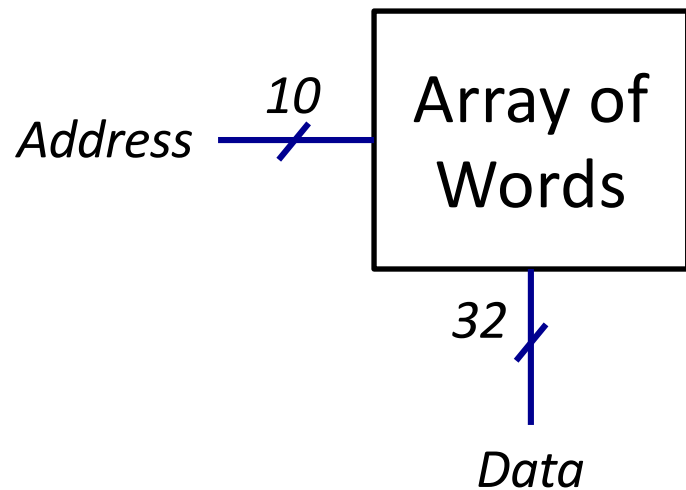
- Data is stored inside memory like the register file
- Address is presented to memory by an external circuit



- Each house is a memory cell (contains data/bit)
- Can reach house if address is known (address is not stored)
- Individual cells (bits) are not addressable
- Typically, a group of 8 bits (byte) has a unique address

Example

- Size of memory (left) =
- Size of memory (right) =

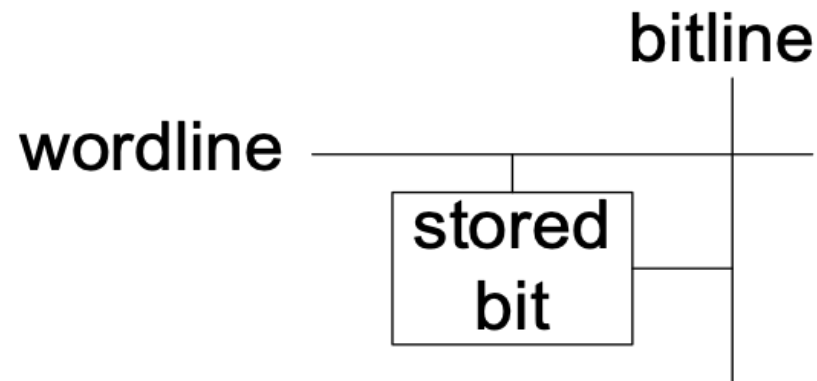


Read/Write Access

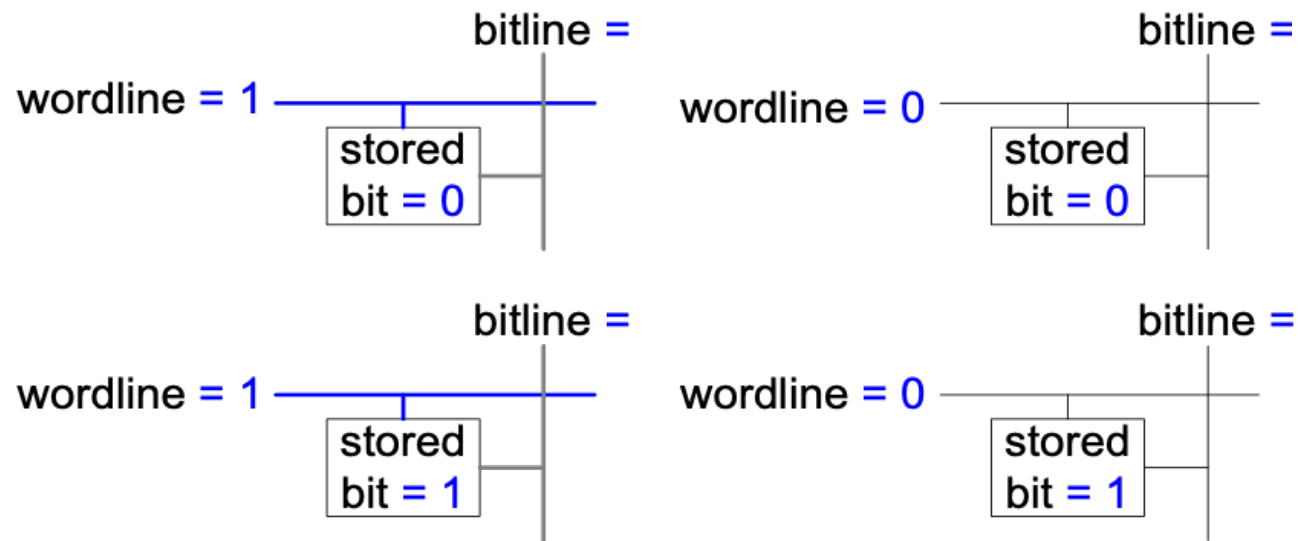
- Looking up a word stored at an address in memory is called a *read access* or simply *read*
- Updating a word stored at an address in memory is called a *write access* or simply *write*
- Typically, a read is called a *memory load* or simply *load* when the word is read from memory into register file
- Similarly, write is called a *memory store* or simply *store*

Memory Cell

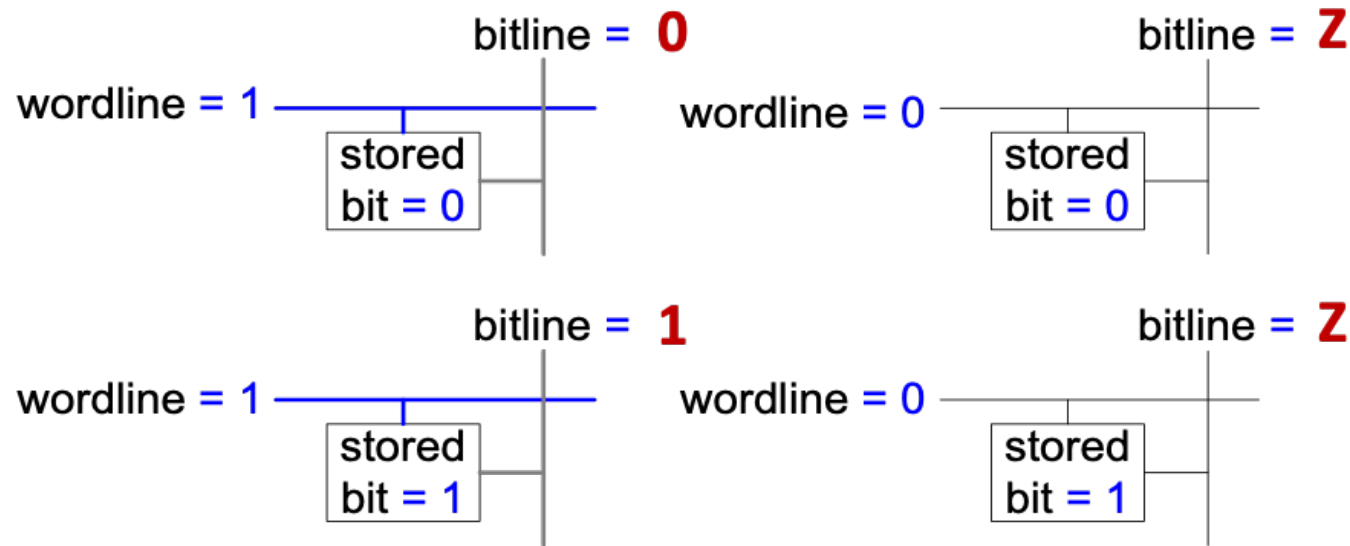
- We call it a bit cell (more technical term)
- A bit cell is connected to a bitline and a wordline
- Each bit cell contains one bit of data
- When the wordline is **HIGH**, the stored bit transfers to or from the bitline



Example: Bit Cell Operation



Example: Bit Cell Operation



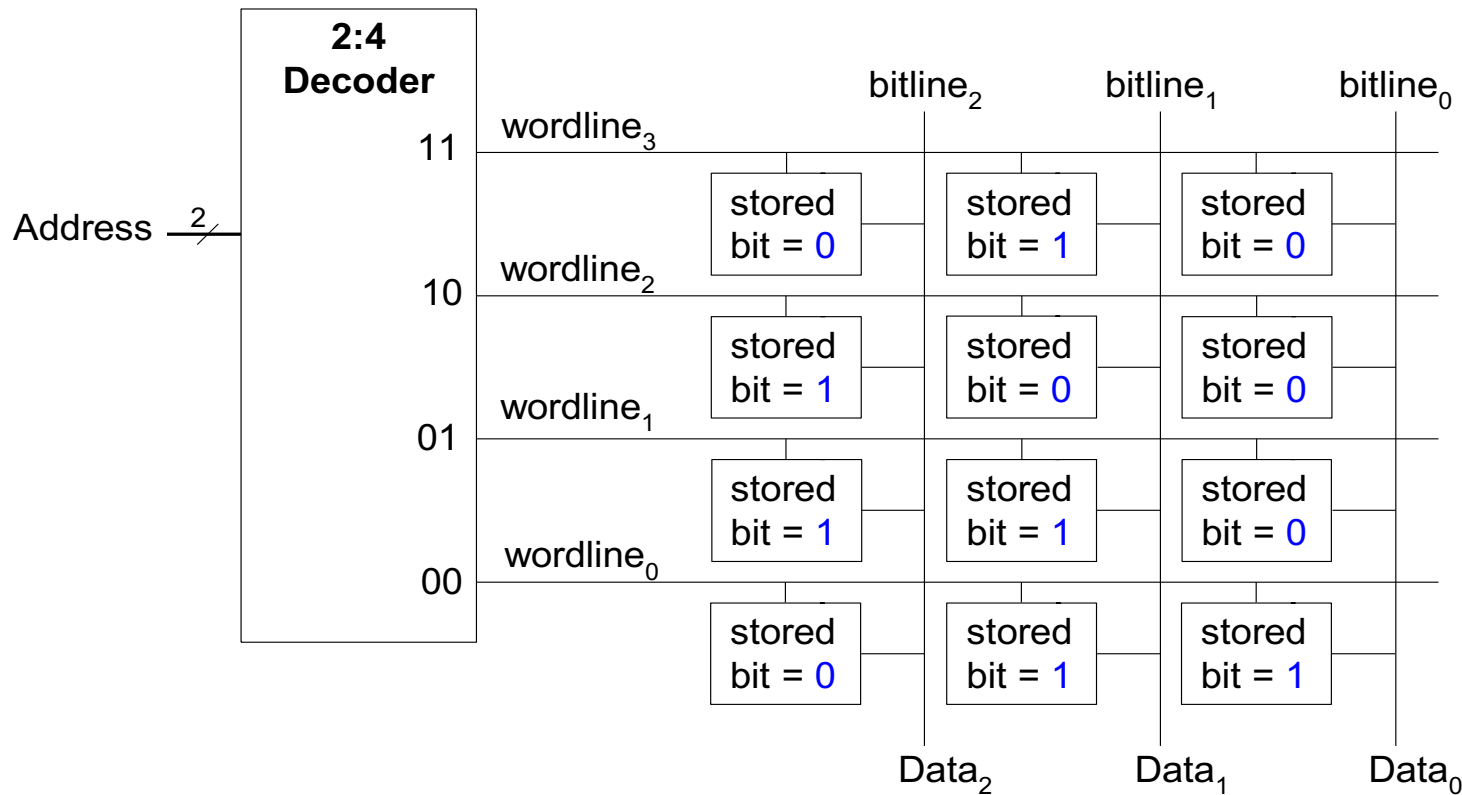
- Z is neither 0 nor 1 in digital electronics
- The wire is cut-off from the circuit (in this case the bit cell)

Reading and Writing Bit Cell

- Read
 - A special circuit is used to bring the bitline in Z state
 - The wordline is then set to HIGH
 - The stored value in the bit cell then drives the bitline
- Write
 - The bitline is driven (set) to 0 or 1
 - The wordline is turned on, connecting the bitline to the stored bit
 - The contents of the bit cell change to 0 or 1

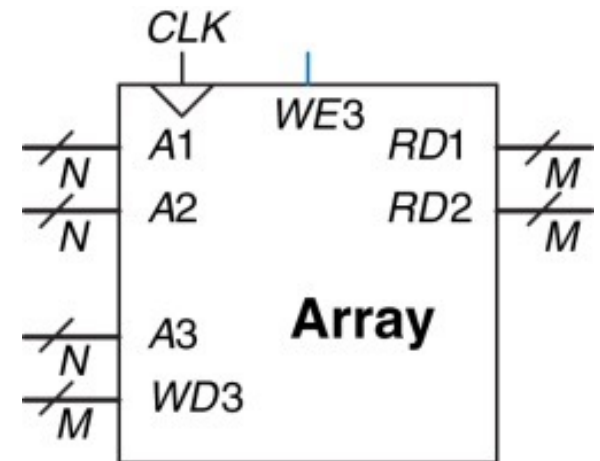
Memory Array Organization

- Decoder drives the wordline **HIGH** based on the address
- Data on the selected row appears on the bitlines



Memory Ports

- Each memory port gives read and/or write access to one memory address
- Multiported memories can access several address simultaneously
- Example of three-ported memory
 - **Port 1** reads the data from address **A1** onto the read data output **RD1**
 - **Port 2** reads the data from address **A2** onto the read data output **RD2**
 - **Port 3** writes the data from the write data input **WD3** into address **A3** on the rising clock edge if **WE3** is **HIGH**



Memory Specification

- Size
 - Width
 - Depth
- Ports
 - How many?
 - Type

Random Access Memory

- Random access memory or RAM is a type of memory for which accessing any data word results in the same delay as any other data word
- The main memory in a typical computer (e.g., your laptops) is RAM
- RAM is volatile
 - If the power is removed from the computer, the data in RAM is no longer there

RAM vs. Storage



- Hard disk (left) and tape (right)
 - Sequential access is faster than random access
 - Mechanical movement is required to access data
 - Non-volatile or persistent storage

Memory Classification

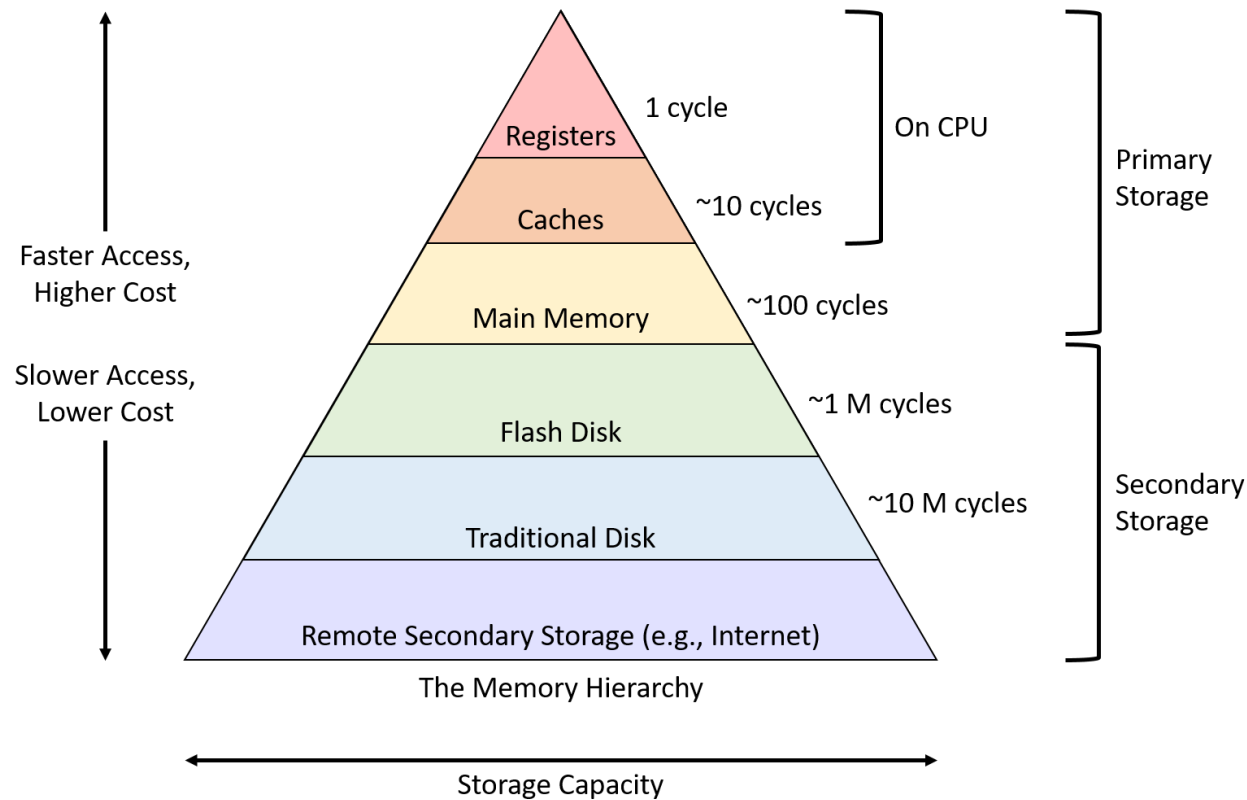
- RAM is classified according to the formation of the bit cells
 - **Static RAM** stores data bits using a pair of cross-coupled inverters
 - **Dynamic RAM** stores data bits using the presence or absence of charge on a capacitor

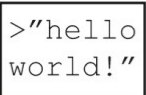


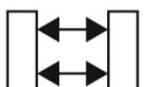
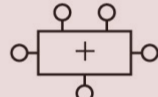
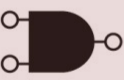
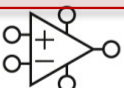


SRAM vs DRAM

- SRAM is fast and expensive
- Typically, the memory close to the CPU uses the SRAM technology
 - Register file
 - Cache (*after the teaching break*)
- The memory far from the processor uses the DRAM technology
 - Your computer's main memory is DRAM

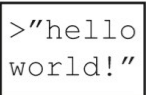



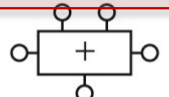
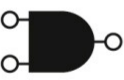
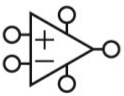


Memory Hierarchy

- We will return to this stuff after the teaching break



Application Software		Programs
Operating Systems		Device Drivers
Architecture		Instructions Registers
Micro-architecture		Datapaths Controllers
Logic		Adders Memories
Digital Circuits		AND Gates NOT Gates
Analog Circuits		Amplifiers Filters
Devices		Transistors Diodes
Physics		Electrons

We are here

Application Software		Programs
Operating Systems		Device Drivers
Architecture		Instructions Registers
Micro-architecture		Datapaths Controllers
Logic		Adders Memories
Digital Circuits		AND Gates NOT Gates
Analog Circuits		Amplifiers Filters
Devices		Transistors Diodes
Physics		Electrons

Moving up a few abstraction layers?

First: Architecture
Then: Microarchitecture

The Architecture Layer

- Our goal in the first half of the course is to understand and build a processor
- We can specify the combinational circuits and the traffic light controller in English
- How should we write the specification of a processor?
 - We need a systematic approach to manage the *complexity* of a processor
- The formal specification of a processor (and computer) takes place at the “architecture” abstraction layer

Architecture/ISA

- The architecture or **I**nstruction **S**et **A**rchitecture (**ISA**) is the programmer's view of the computer
- ISA specifies the set of instructions a computer can perform (think of it like the language of a computer)
 - Instruction = word
 - ISA = vocabulary
- Each instruction specifies
 - Operation (What exactly to do?)
 - Operands (Where to find the data to operate upon?)
- Example: **Add** two 16-bit binary numbers in registers **R1** and **R2** (recall the register file from Lab 3 – 4)

Operands

- Operands can be in register and memory
 - Programs need more than a few registers
 - Register file is small and expensive
- If operands can be in memory, then we must have instructions:
 - To *Load* from memory into registers
 - To *Store* from registers to memory
- Is there a third possibility for operand location?
 - Encoded in the instruction as 1's and 0's

Assembly Language

- Instructions written in a symbolic format so humans can read/understand them easily is called assembly language
 - ADD, SUB, LDR, STR, MUL, ROR, MOV, BIC
- A sequence of instructions is called assembly code

*Remark: Don't worry
about what this means!
Just want to show how
assembly looks*

SUB	R0,	R1,	R2
ADD	R8,	R4,	R5
ADD	R9,	R6,	R7
SUB	R3,	R8,	R9

Machine Language

- Instructions are encoded as 1's and 0's in a format called the machine language
 - ADD can be represented by binary code 0000
 - SUB can have a possible encoding of 0001
 - Register R1: 00 (example)
 - Register R2: 01

```
0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0
0 0 0 1 1 0 0 0 0 0 1 1 1 0 0 0
```

Hypothetical machine code (above) is a sequence of machine language instructions stored in memory

Instruction Format

- An instruction consists of several fields
- Each field has a different meaning
- An instruction format specifies the meaning of each field
- An ISA can have many instruction formats

Microarchitecture

- An ISA is a specification
- Microarchitecture is the implementation of an ISA
 - The specific arrangement of registers, memories, ALUs, and other building blocks to form a processor is called microarchitecture
- The same ISA can have many different implementations
 - Tradeoffs in performance, power, price
 - A company X builds two processors for a high-end laptop and a cheap cell phone, respectively, that can both run programs targeting the same ISA named Y

More Examples

- Intel and AMD build processors targeting the x86 ISA
- QuAC ISA is for teaching purposes
 - Each group/student will build a CPU differently
 - One group may use two adders rather than one; different styles for register file
 - Both are building a processor for the same ISA
 - Their microarchitectures are different

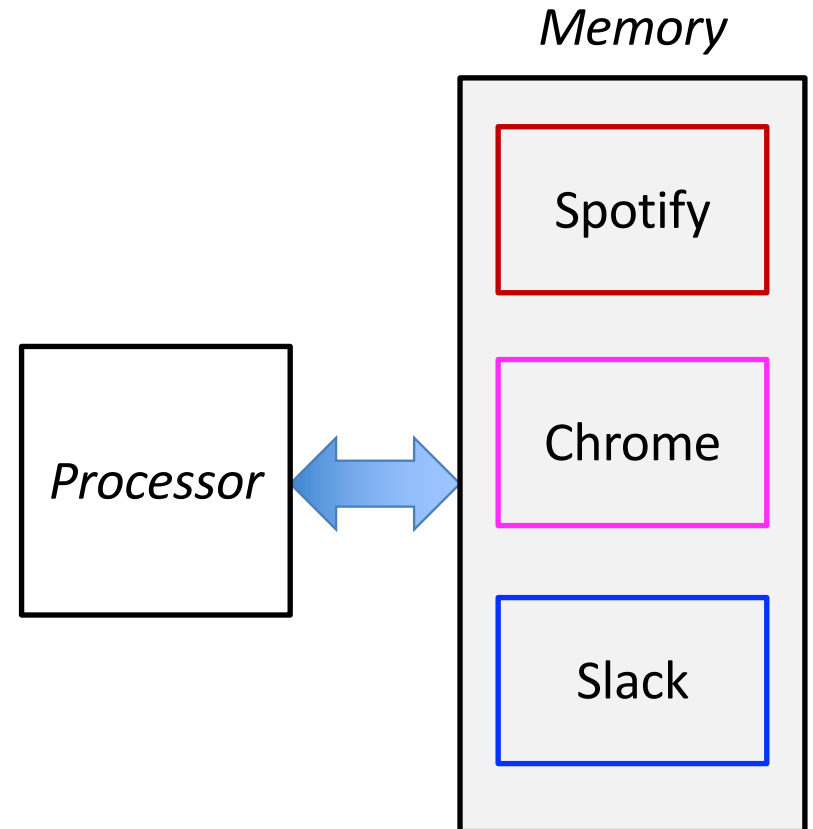


Some Remarks

- All programs running on a computer use the same instruction set
- All software applications, such as Spotify and Word, are eventually *compiled* into a series of simple instructions
- **Compiler:** A program that transforms a program written in a high-level language (C, C++, Python) to assembly instructions
- **Assembler:** A program that transforms assembly code to machine code

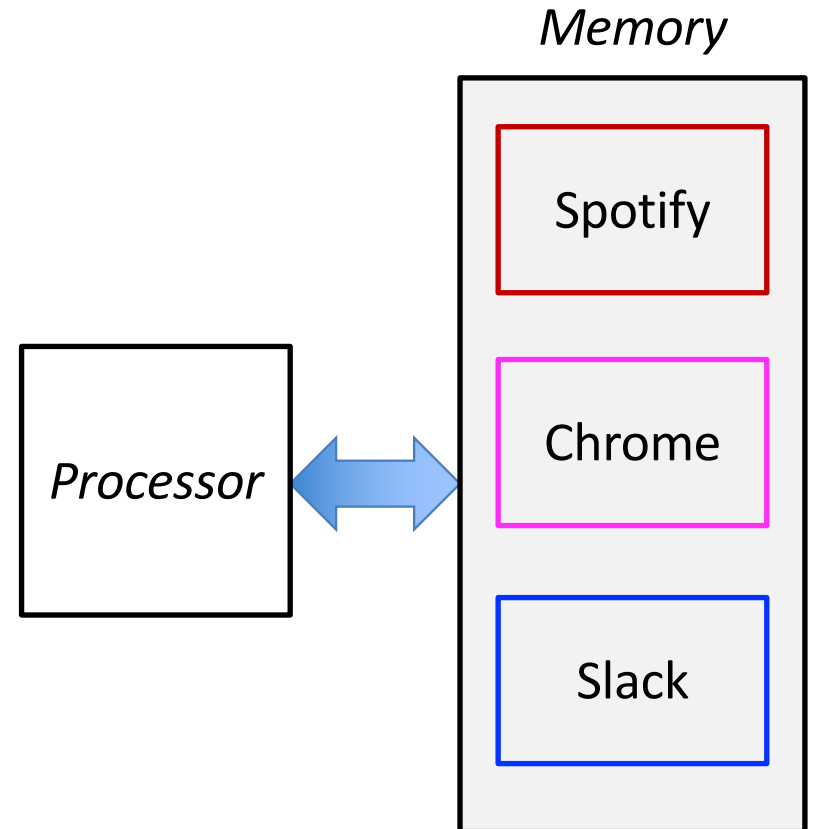
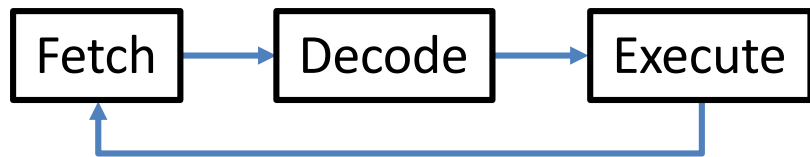
Stored Program Concept

- Two key principles
 - Instructions are represented as binary numbers
 - Programs (*in the form of machine code*) are stored in memory (*like data*)



Stored Program Concept

- How do processors execute machine code?
 - Fetch an **instruction** from memory
 - Decode the meaning of the **instruction**
 - Execute the **instruction**
 - Repeat until out of instructions



Popular ISAs

- Intel x86
 - High-performance desktop/laptop/server
 - Power-hungry (Apple's recent shift to Apple silicon)
- ARM
 - Popular in the mobile/embedded domain
 - Lecture/textbook focus
 - M1 (Apple) uses ARM architecture
- RISC-V
 - A new open-source ISA gaining momentum
- QuAC
 - Teaching purposes (invented at ANU)

Instruction Widths

- Intel x86
 - Instructions are *variable-sized* (From one up to many bytes)
 - Difficult to implement
- ARM, RISC-V, and QuAC
 - These ISAs have fixed-width instructions
 - ARM has 16-bit, 32-bit, and 64-bit variants
 - QuAC only has 16-bit instructions

Quiz

- Ben has an excellent idea to make computers more efficient. To try out his idea he first picks a popular existing ISA.
 - *What is the advantage of Ben's approach to pick an existing ISA instead of developing a new one?*
 - *What is the drawback of picking an existing ISA?*

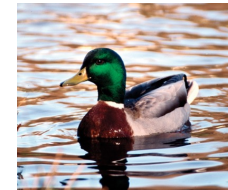
Quiz

- Ben has an excellent idea to make computers more efficient. To try out his idea he first picks a popular existing ISA.
 - *What is the advantage of Ben's approach to pick an existing ISA instead of developing a new one?*
 - *Binary compatibility enables existing programs to make use of Ben's excellent idea without any effort*
 - *Historically, this aspect has led to **ISA hegemony**, where one popular ISA is dominant*
 - *What is the drawback of picking an existing ISA?*
 - *Think!*

ISA Choice



- We will study the **ARMv4** 32-bit ISA
 - Billions of devices (cell phones, laptops) use ARM ISA
- ISA choices impact microarchitecture complexity
 - ARM is known for low-power implementations
- The lab/assignment ISA (**QuAC**) is a 16-bit ISA
 - Implementing full ARM ISA is a multi-year effort
 - For teaching purposes, we designed QuAC



Remark: Even in lectures, we will first study a subset of ARM, and see its implementation, before moving on to more advanced features

Instruction Classes

What are the fundamental things a computer must do?

1. *Data processing instructions (arithmetic/logical instructions)*
2. *Data movement instructions (memory accesses)*
3. *Decision making instructions (if/else)*
4. *Function calls (modularity/structured programming)*

Data Processing Instructions

- Suppose I have a statement in C language
 - $a = b + c - d$

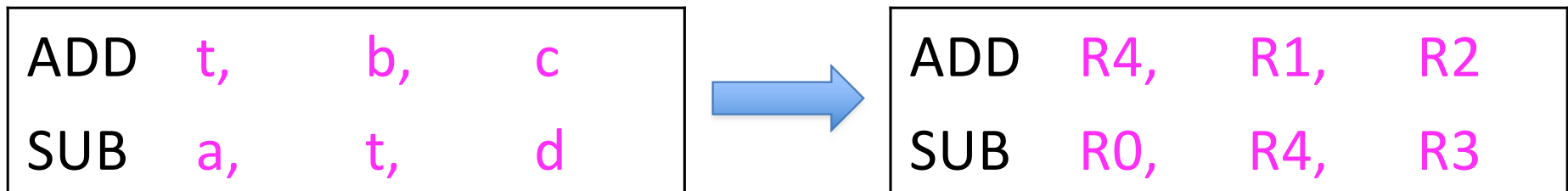
ARM Assembly Code
two instructions

ADD	t,	b,	c
SUB	a,	t,	d

- ADD and SUB are instruction mnemonics
- Instructions operate on operands
 - a, b, c are operands
- Computers operate on binary data and not variable names
- We need to specify the physical location of operands
 - Registers, memory, constants in instructions

Registers as Operands

- Instructions need fast access to operands, but memory is slow
 - Keep a small set of registers close to the CPU in a register file
 - ARM architecture uses 16 registers
 - 32-bit architecture means 32-bit registers
- Let's assume the following mapping of variables to registers in the C statement, **a = b + c - d**
 - $R0 = a, R1 = b, R2 = c, R3 = d, R4 = t$



Source/Destination Operand

- Instructions operate on one or two source operands and store the result after execution in a destination operand
- R1 and R2 are the *source operands* for the ADD instruction
- R4 is the *destination operand* for the ADD instruction

ADD	R4,	R1,	R2
SUB	R0,	R4,	R3

Exercise

- Translate the following C code into ARM assembly. Assume variables $a - c$ are held in registers $R0 - R2$ and $f - j$ are held in registers $R3 - R7$
 - $a = b - c$
 - $f = (g + h) - (i + j)$

```
SUB  R0,  R1,  R2
ADD  R8,  R4,  R5
ADD  R9,  R6,  R7
SUB  R3,  R8,  R9
```

The Register Set

- ARM defines 16 *architectural* registers
- The register set is part of the ISA specification
- R0 – R12 are used for storing variables
- R13 – R15 have special uses

Constants/Immediates

- ARM instructions can use constant or immediate operands
- The value is available immediately from the instruction
 - No register or memory access
 - Immediates can be 8 – 12 bits (reason?)

In the following example, assume $R7 = a$, $R8 = b$

C code:

$a = a + 4$

$b = a - 12$

ARM Assembly Code

ADD	R7,	R7,	#4
SUB	R8,	R7,	#0xC

MOV Instruction

- MOV is a useful instruction for initializing register values
- MOV can also take a register source operand
 - MOV R1, R7 copies the contents of register R7 into R1

In the following example, assume R4 = i, R5 = x

C code:

```
i = 0;  
x = 4080;
```

ARM Assembly Code

MOV	R4,	#0
MOV	R5,	#0xFF0

More Data Processing Insts.

- AND
- ORR (OR)
- EOR (XOR)
- BIC (Bit Clear)
- MVN (MoVe and Not)

The Bit Clear Instruction

- Bit Clear (BIC)
 - Masking bits (forcing unwanted bits to 0)
- BIC R6, R1, R2
 - R2 is the mask (the bits we want to clear or zero in R1 are set to 1 in R2)
 - The instruction stores the result of R1 AND (NOT R2) in R6

Example: Data Processing

Source registers

R1	0100 0110	1010 0001	1111 0001	1011 0111
R2	1111 1111	1111 1111	0000 0000	0000 0000

Assembly code

AND R3, R1, R2
ORR R4, R1, R2
EOR R5, R1, R2
BIC R6, R1, R2
MVN R7, R2

Result

R3	0100 0110	1010 0001	0000 0000	0000 0000
R4	1111 1111	1111 1111	1111 0001	1011 0111
R5	1011 1001	0101 1110	1111 0001	1011 0111
R6	0000 0000	0000 0000	1111 0001	1011 0111
R7	0000 0000	0000 0000	1111 1111	1111 1111

Data Processing Instructions

- Problem
 - Programs need more than 16 registers
 - Memory is large but slow
- Solution
 - Use both register file and memory
 - Use registers whenever possible
- Two **instructions** to facilitate data movement
 - The **LDR** instruction: Bring data words from memory into the register file
 - The **STR** instruction: Store data words from the register file to memory

Memory Organization

ISA specifies the smallest unit of memory the CPU can access

All ISAs (including ARM) use byte-addressable memory

- Memory is organized into bytes
- Each byte has a unique address starting from 0 to $2^N - 1$

QuAC ISA (teaching purposes) uses word-addressable memory

- Memory is organized as 16-bit words
- Each word has a unique address starting from 0 to $2^{N-1} - 1$

Memory View (32 bits = 4 bytes)

Byte-addressable memory (each box is a byte; each row is a word)
Byte addresses (**left**) and 8-bit byte data (**right, 1 byte = 2 Hex digits**)

Byte Address				Word Address	Data				Word Number
⋮				⋮	⋮				⋮
13	12	11	10	00000010	CD	19	A6	5B	Word 4
F	E	D	C	0000000C	40	F3	07	88	Word 3
B	A	9	8	00000008	01	EE	28	42	Word 2
7	6	5	4	00000004	F2	F1	AC	07	Word 1
3	2	1	0	00000000	AB	CD	EF	78	Word 0
MSB		LSB			← 4 Bytes →				

Reading from Memory

- Format of Load Register instruction
`LDR R0, [R1, #12]`
- Address calculation
 - Add base address (R1) to the offset (12)
 - $\text{Address} = (R1 + 12)$
 - Use any register for base address
 - R1 is a source (register) operand
- Result
 - R0 holds the data at memory address $(R1 + 12)$ after the instruction executes
 - R0 is a destination (register) operand

LDR Example

Read a 32-bit word of data at memory (byte) address 8 into R3. Use R2 as the base register. Show the contents of R3.

- Let's initialize R2 to 0, and add 8 as the offset

```
MOV R2, #0
LDR R3, [R2, #8]
```

R3	0x01EE2842
----	------------

Word Address	Data				Word Number
⋮	⋮				⋮
00000010	CD	19	A6	5B	Word 4
0000000C	40	F3	07	88	Word 3
00000008	01	EE	28	42	Word 2
00000004	F2	F1	AC	07	Word 1
00000000	AB	CD	EF	78	Word 0

Writing to Memory

- Format of **STore Register** instruction
STR R0, [R1, #12]
- Address calculation
 - Add base address (R1) to the offset (12)
 - $\text{Address} = (R1 + 12)$
 - R0 and R1 are both source (register) operands
- Result
 - Memory address (R1 + 12) contains the value in R0 after the instruction executes
 - The second operand is the destination, or the destination operand is memory address

STR Example

Store the value held in R7 into memory word 21.

- Let's initialize R5 to 0, and add 84 (21 X 4) as the offset

```
MOV  R5,    #0
STR   R7,    [R5, #0x54]
```

*The offset can be written in decimal or hexadecimal
84 (decimal) is 0x54 (Hex)*