

Instructions – Language of the Computer

Computer Systems Organization, Spring 2017

Lavanya Ramapantulu

Email: lavanya.r@iiit.ac.in

Office: Vindhya A5-304b

Learning Objectives

- ISA – What and Why
- ISA design philosophies: RISC versus CISC
- Execution flow example
- ISA concepts
 - Data storage
 - Memory addressing modes
 - Operations
 - Instruction formats
 - Encoding

Program

- Program = ??
- Algorithm + Data Structures – Niklaus Wirth
- Program (Abstraction) of processor/hardware that executes

Recap: Program Abstractions

High-level
language
program
(in C)

```
swap(int v[], int k)
{int temp;
  temp = v[k];
  v[k] = v[k+1];
  v[k+1] = temp;
}
```

Compiler

Assembly
language
program
(for RISC-V)

```
swap:
    slli x6, x11, 3
    add  x6, x10, x6
    ld   x5, 0(x6)
    ld   x7, 8(x6)
    sd   x7, 0(x6)
    sd   x5, 8(x6)
    jalr x0, 0(x1)
```

Assembler

Binary machine
language
program
(for RISC-V)

```
00000000001101011001001100010011
000000000011001010000001100110011
00000000000000110011001010000011
00000000100000110011001110000011
00000000011100110011000000100011
00000000010100110011010000100011
0000000000000000100000001100111
```

- You write programs in high level programming languages, e.g., C/C++, Java:

k = k+1

- Compiler translates this into assembly language statement:

add x6, x10, x6

- Assembler translates this statement into machine language instructions that the processor can execute:

0000 0000 0110 0101 0000 0011 0011 0011

Program Abstractions – Why ?

- **Machine code**
 - Instructions are represented in binary
 - **1000110010100000** is an instruction that tells one computer to add two numbers
 - Hard and tedious for programmer
- **Assembly language**
 - Symbolic version of machine code
 - Human readable
 - **add A, B** is equivalent to **1000110010100000**
 - **Assembler** translates from assembly language to machine code
- How many assembly instructions should you write to read two memory locations, and add them ?

Instruction Set Architecture

An abstraction on the interface between the hardware and the low-level software.

Software

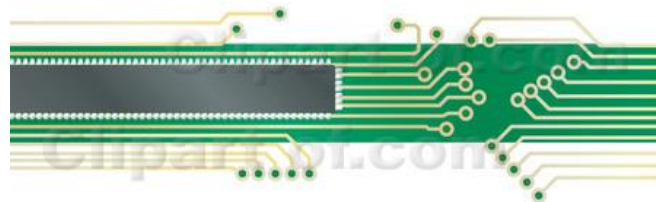
(to be translated to the instruction set)



Instruction Set Architecture

Hardware

(implementing the instruction set)



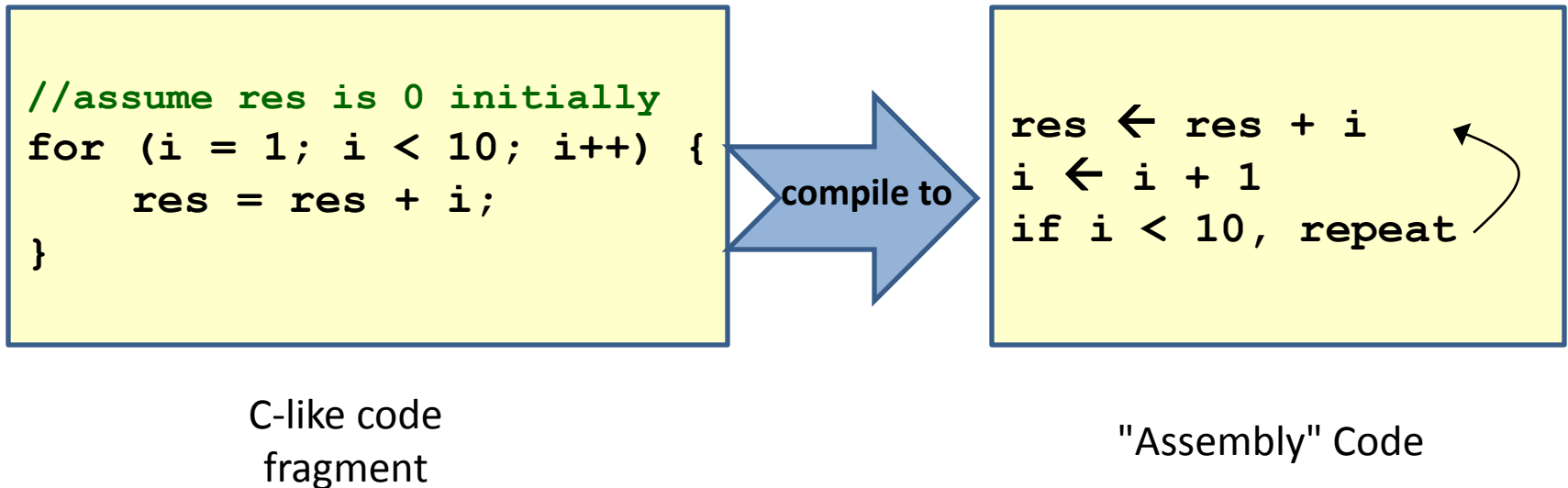
Does a given ISA have a fixed implementation ?

ISA Design Philosophies

- **Complex Instruction Set Computer (CISC)**
 - example: x86-32 (IA32)
 - single instruction performs complex operation
 - smaller assembly code size
 - lesser memory for storing program
- **Reduced Instruction Set Computer (RISC)**
 - example: MIPS, ARM, **RISC-V**
 - multiple instructions to perform complex operation
- Compiler design ?
- Hardware implementation ?

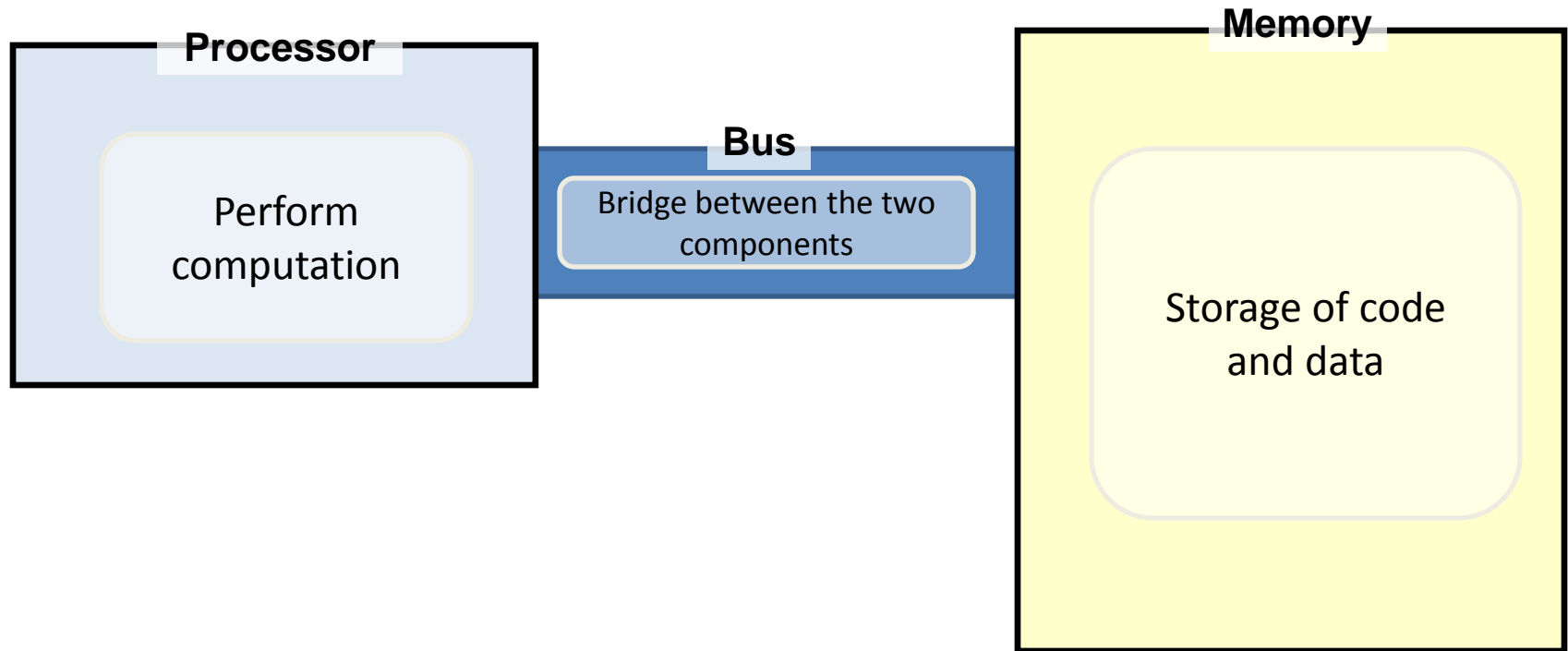
Execution flow of a program

- example of the computer components activated, instructions executed and data flow during an example code execution



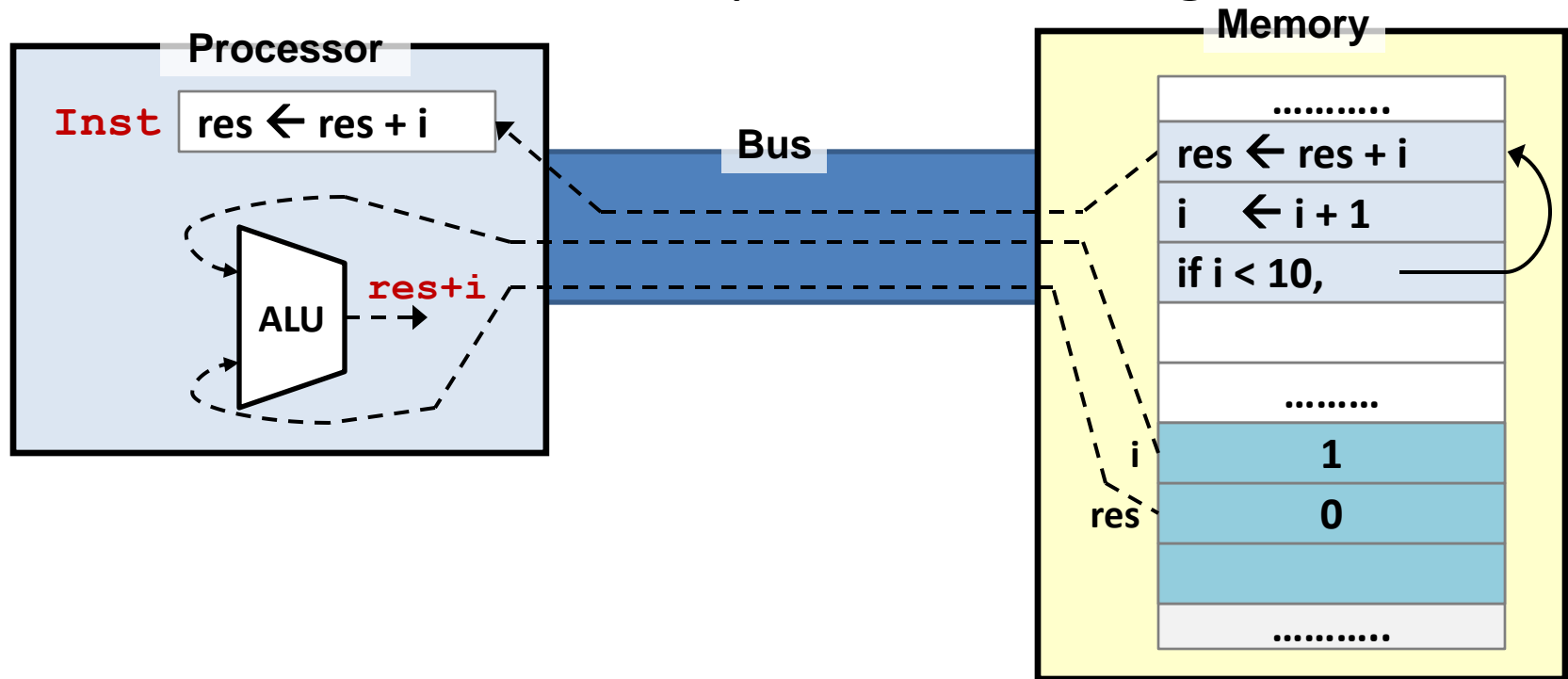
Recap: Computer Components

- What are the two major components in a computer



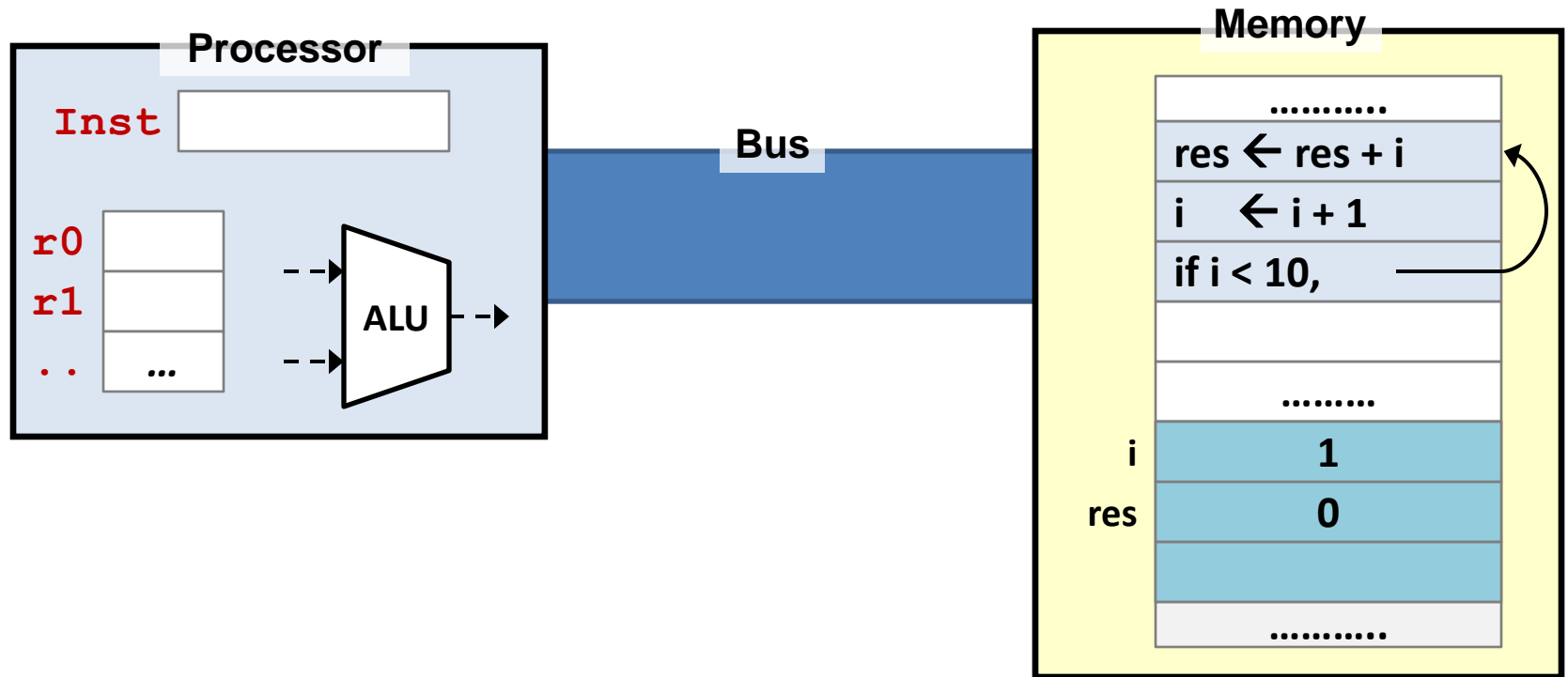
Recap: Execution flow and Data flow

- The code and data reside in memory
 - Transferred into the processor during execution



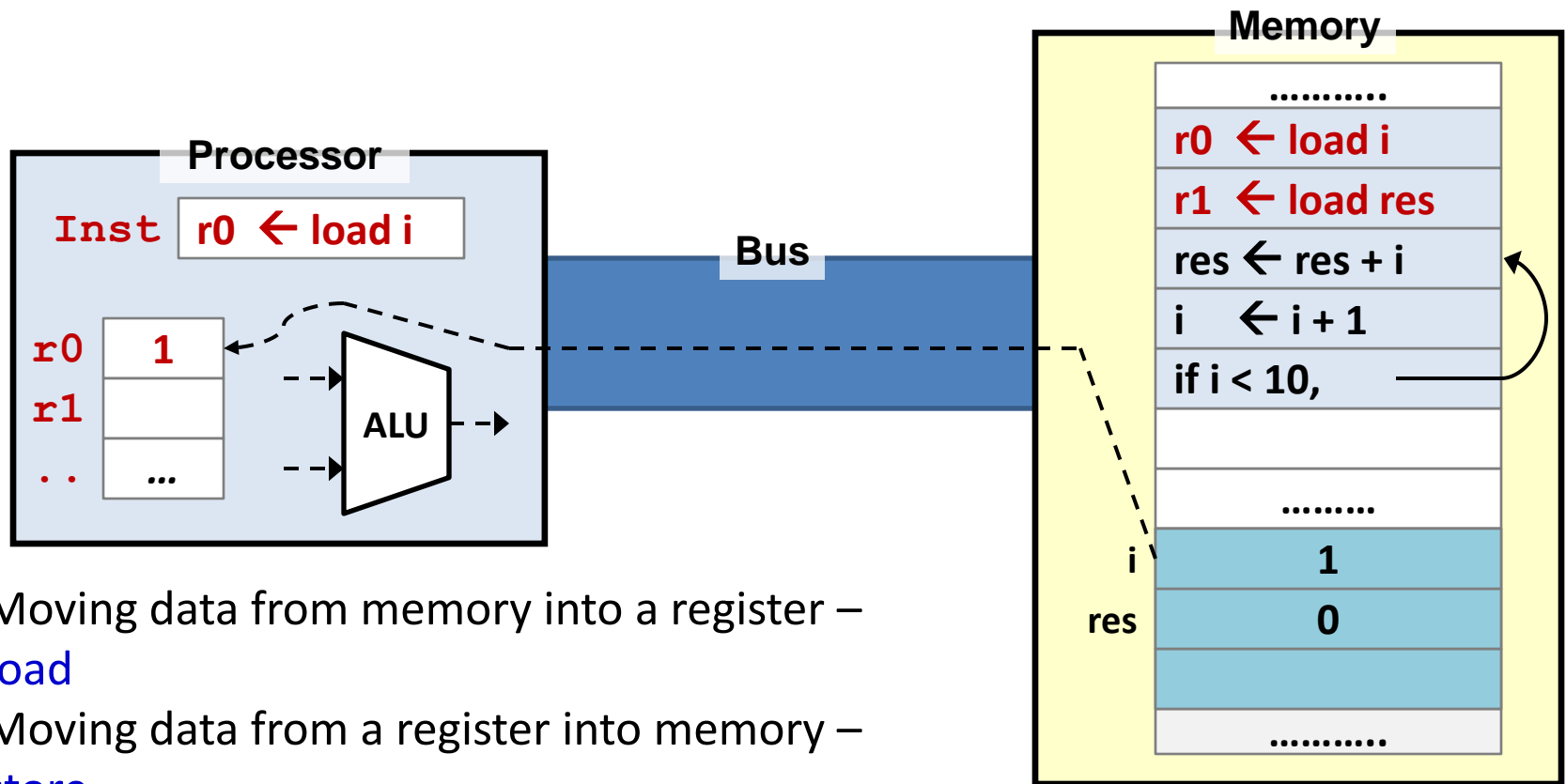
Memory access is slow!

- To avoid frequent access of memory
 - Provide temporary storage for values in the processor (known as **registers**)



Memory instruction

- Need instruction to move data into registers
 - also from registers to memory later



Moving data from memory into a register –

load

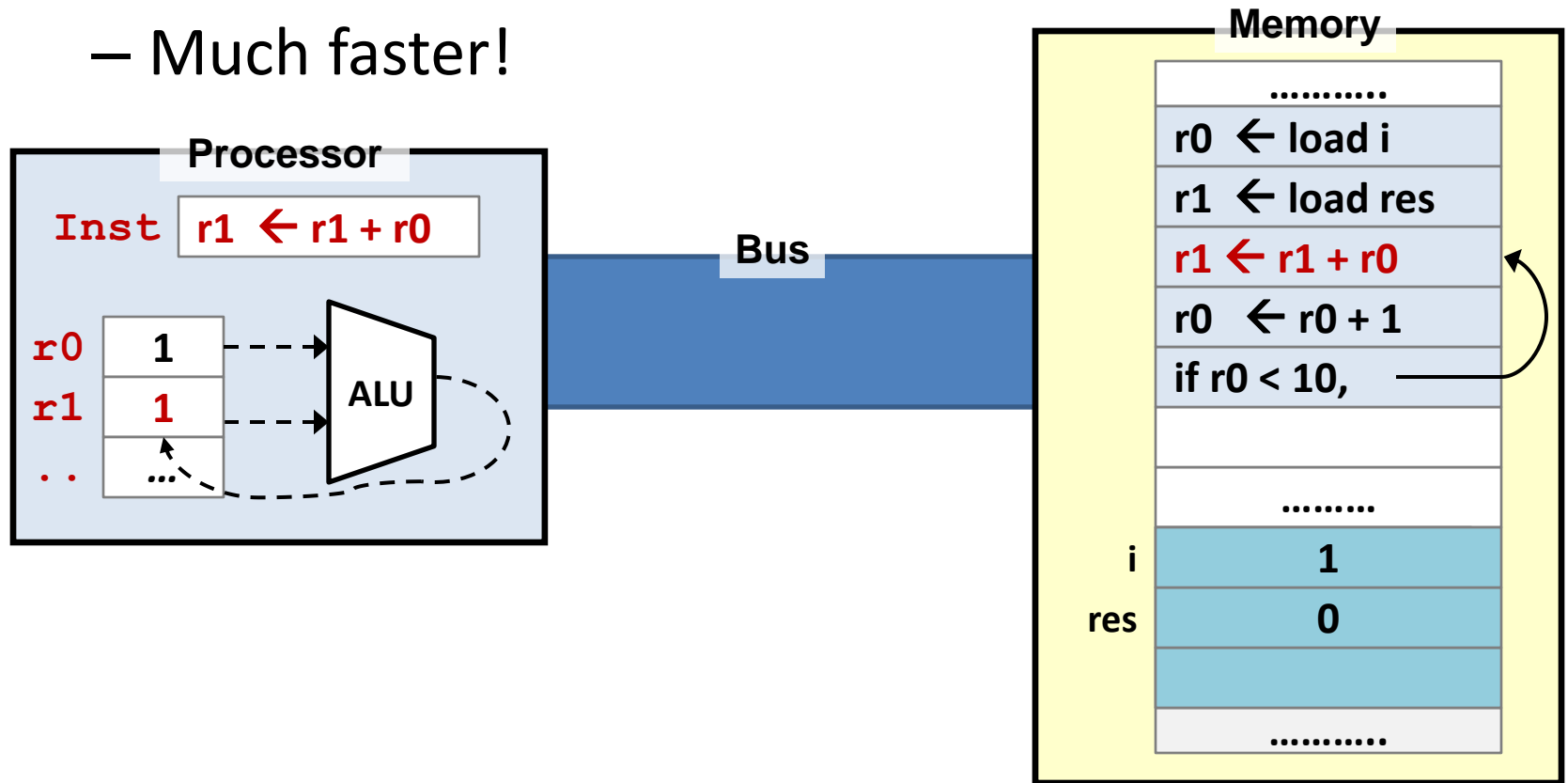
Moving data from a register into memory –

store

Source acknowledgement: <http://www.comp.nus.edu.sg/~cs2100/lect/cs2100-9-MIPS-1-full.pptx>

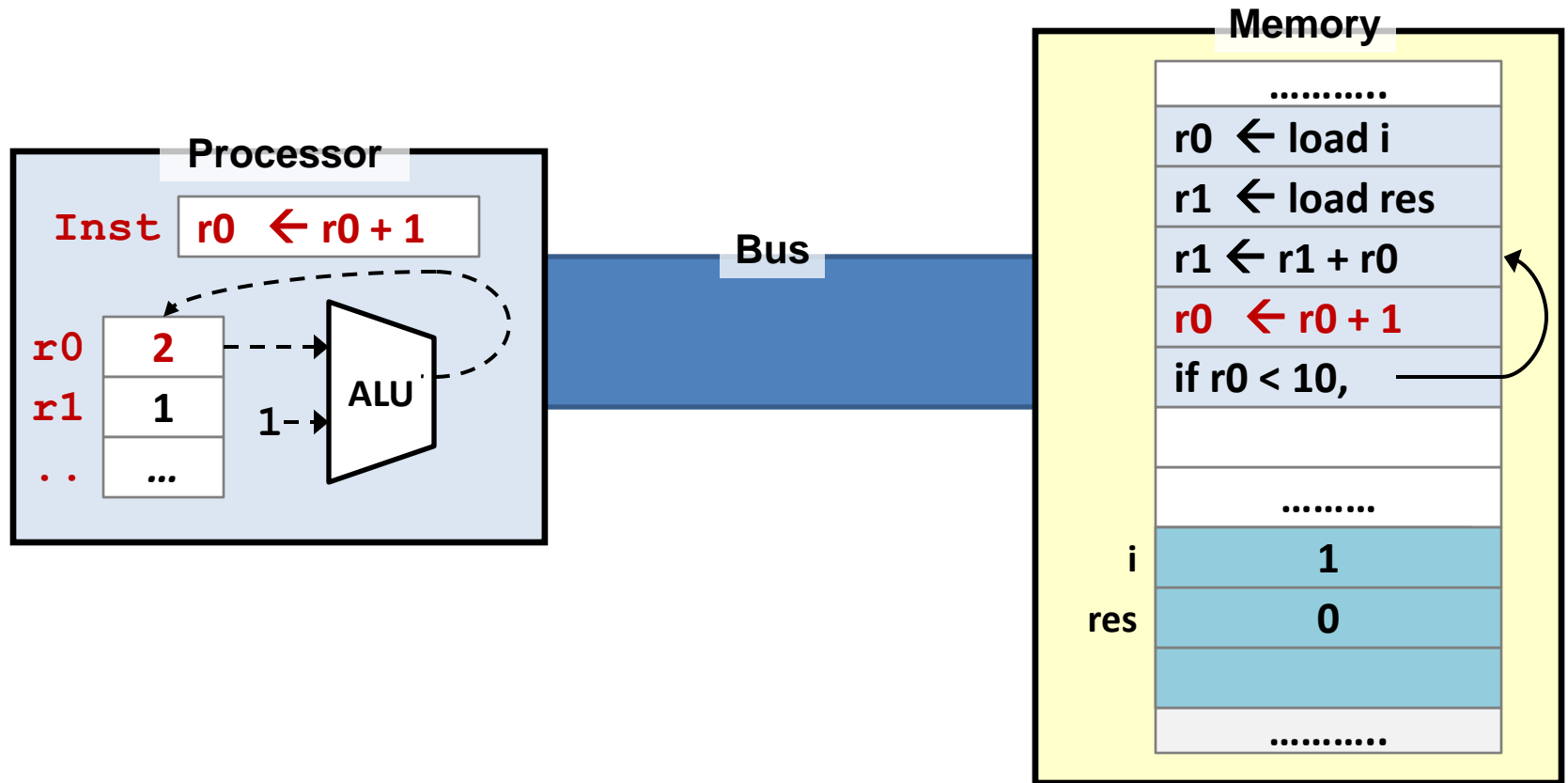
Reg-to-Reg Arithmetic

- Arithmetic operation can now work directly on registers only:
 - Much faster!



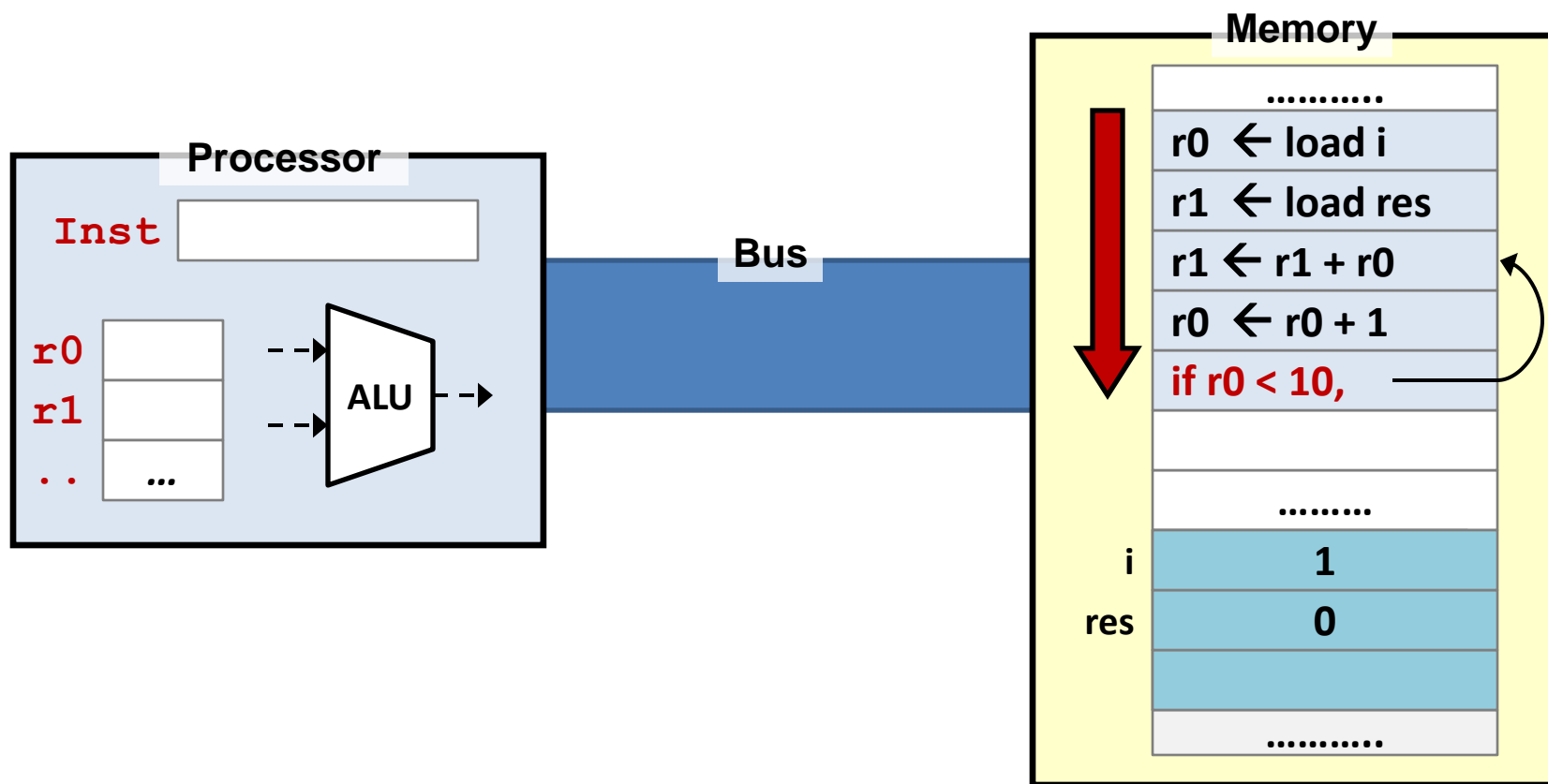
Reg-to-Reg Arithmetic

- Sometimes, arithmetic operation uses a **constant** value instead of register value



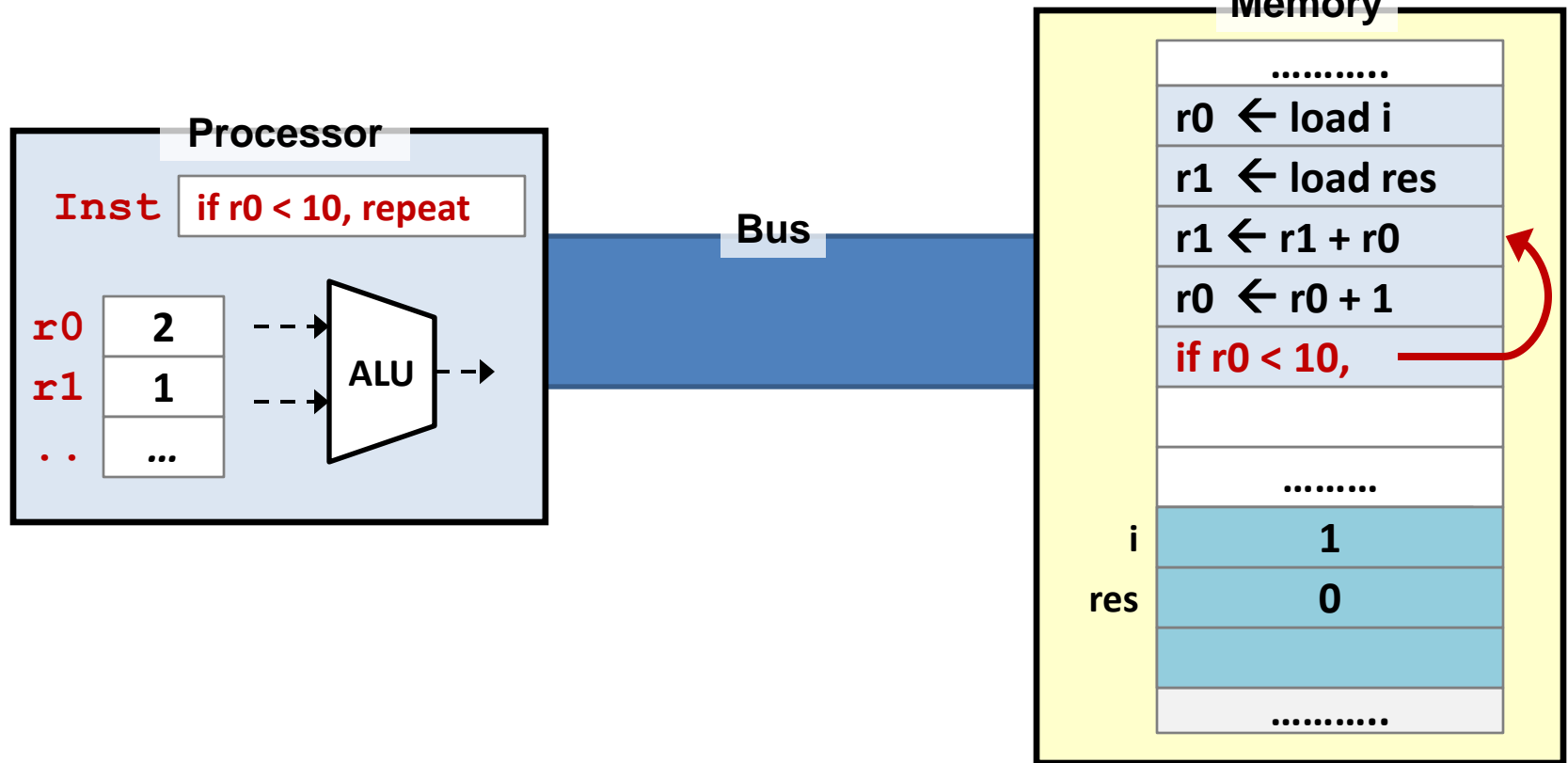
Execution sequence

- Instruction is executed sequentially by default
 - How do we “repeat” or “make a choice”?



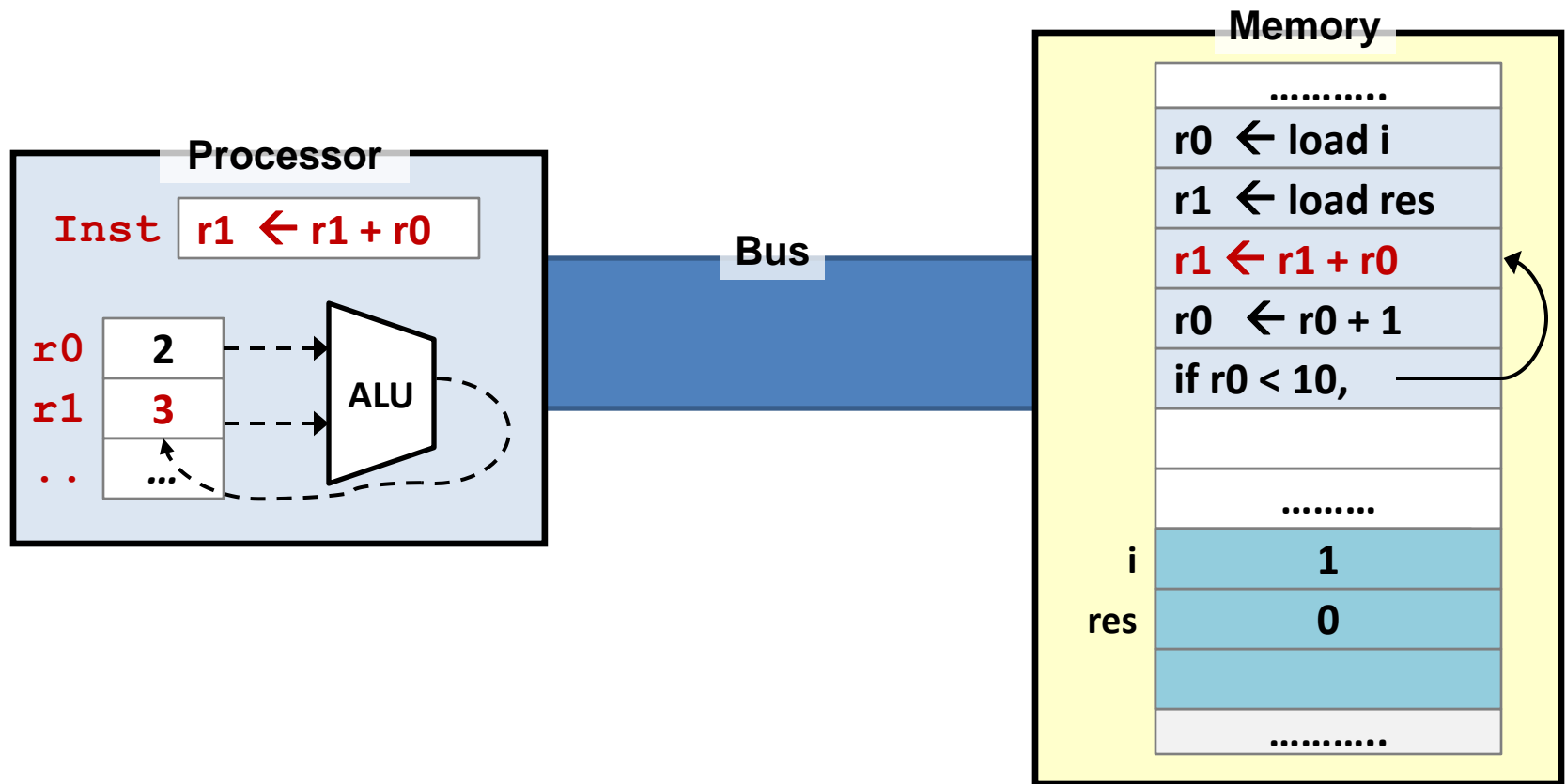
Control flow instruction

- We need instruction to **change** the control flow based on **condition**:
 - Repetition (loop) and Selection (if-else) can both be supported



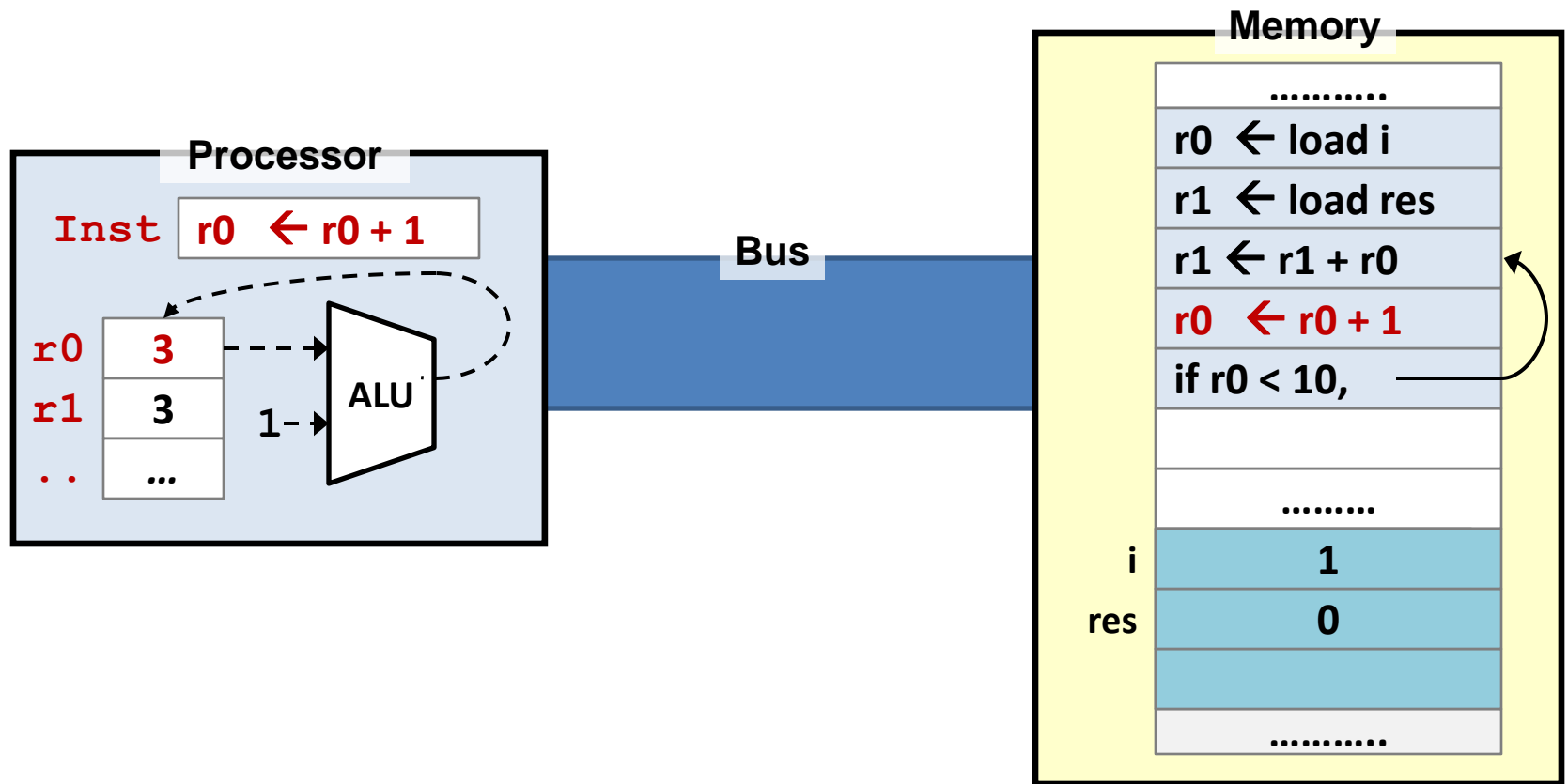
Looping!

- Since the condition succeeded, execution will repeat from the indicated position



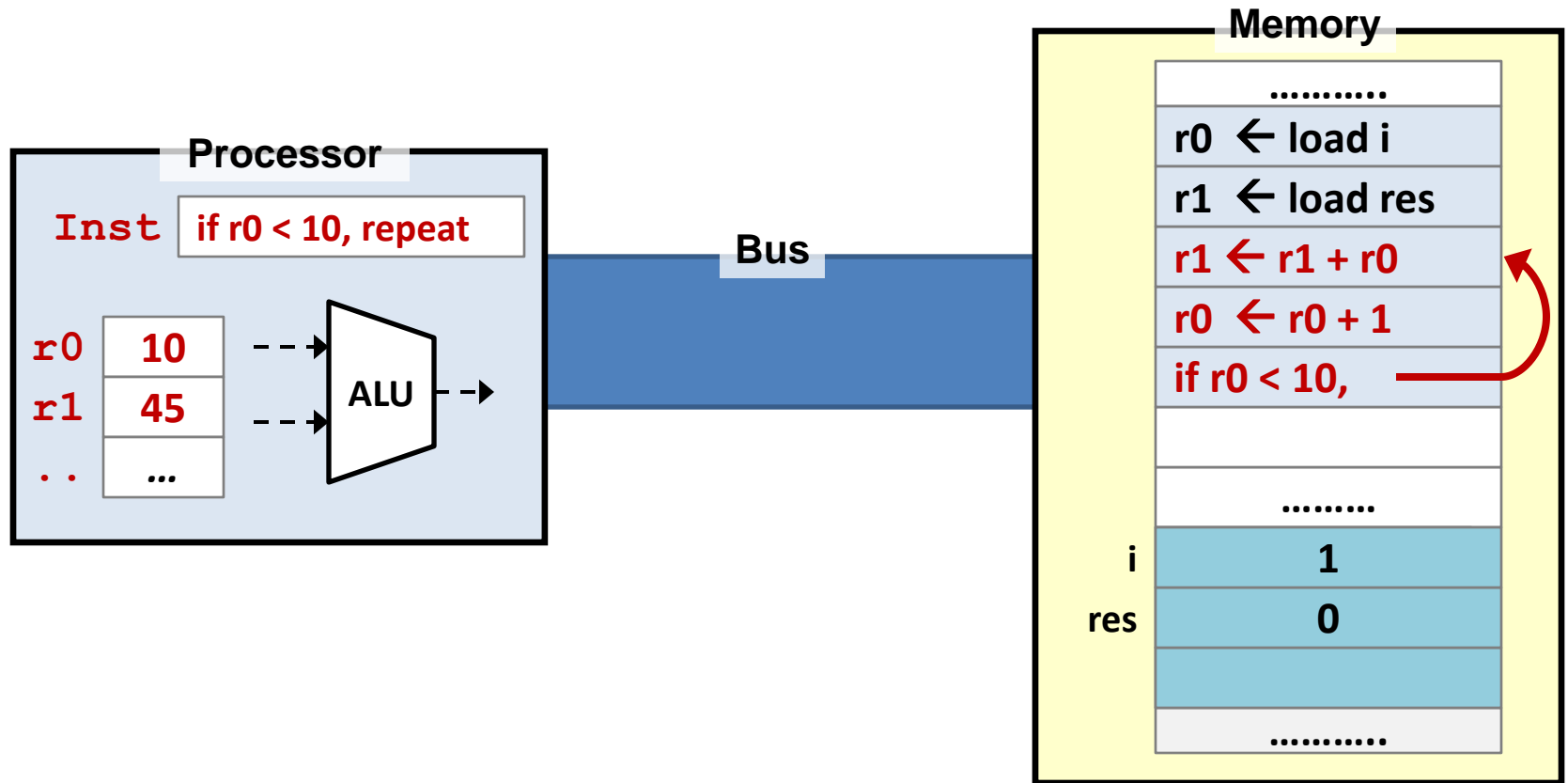
Looping!

- Execution will continue sequentially:
 - Until we see another control flow instruction!



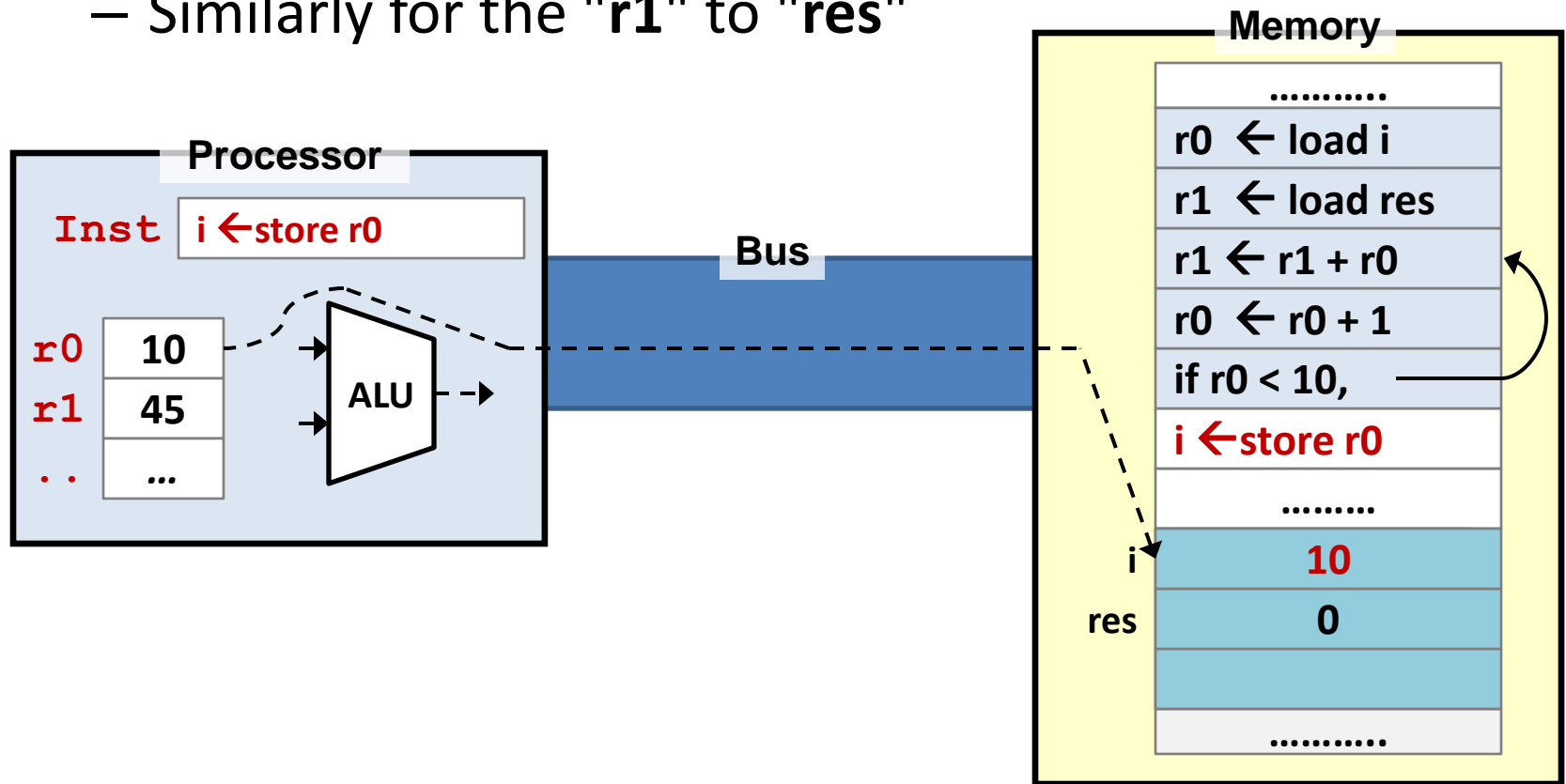
Control flow instruction

- The three instructions will be repeated until the condition fails



Memory instruction

- We can now move back the values from register to their “home” in memory
 - Similarly for the "r1" to "res"



Summary of observations

- The stored-memory concept:
 - Both **instruction** and **data** are stored in memory
- The load-store model:
 - Limit memory operations and relies on registers for storage during execution
- The major types of assembly instruction:
 - **Memory:** Move values between memory and registers
 - **Calculation:** Arithmetic and other operations
 - **Control flow:** Change the sequential execution

ISA Concepts

Concept #1: Data Storage

Concept #2: Memory Addressing Modes

Concept #3: Operations in the Instruction Set

Concept #4: Instruction Formats

Concept #5: Encoding the Instruction Set

Concept #1: Data Storage

- Storage Architecture
- General Purpose Register Architecture

Concept #1: Data Storage

Concept #2: Memory Addressing Modes

Concept #3: Operations in the Instruction Set

Concept #4: Instruction Formats

Concept #5: Encoding the Instruction Set

Source acknowledgement: <http://www.comp.nus.edu.sg/~cs2100/lect/cs2100-13b-ISA.pptx>

Storage Architecture: Definition



- Under Von Neumann Architecture:
 - Data (operands) are stored in memory
- For a processor, **storage architecture** concerns with:
 - Where do we store the operands so that the computation can be performed?
 - Where do we store the computation result afterwards?
 - How do we specify the operands?
- Major storage architectures in the next slide

Source acknowledgement: <http://www.comp.nus.edu.sg/~cs2100/lect/cs2100-13b-ISA.pptx>

Storage Architecture: Common Design

- **Stack architecture:**

- Operands are implicitly on top of the stack.

- **Accumulator architecture:**

- One operand is implicitly in the accumulator (a register). Examples: IBM 701, DEC PDP-8.

- **General-purpose register architecture:**

- Only explicit operands.
- **Register-memory architecture** (one operand in memory). Examples: Motorola 68000, Intel 80386.
- **Register-register (or load-store) architecture.**
Examples: MIPS, DEC Alpha.

- **Memory-memory architecture:**

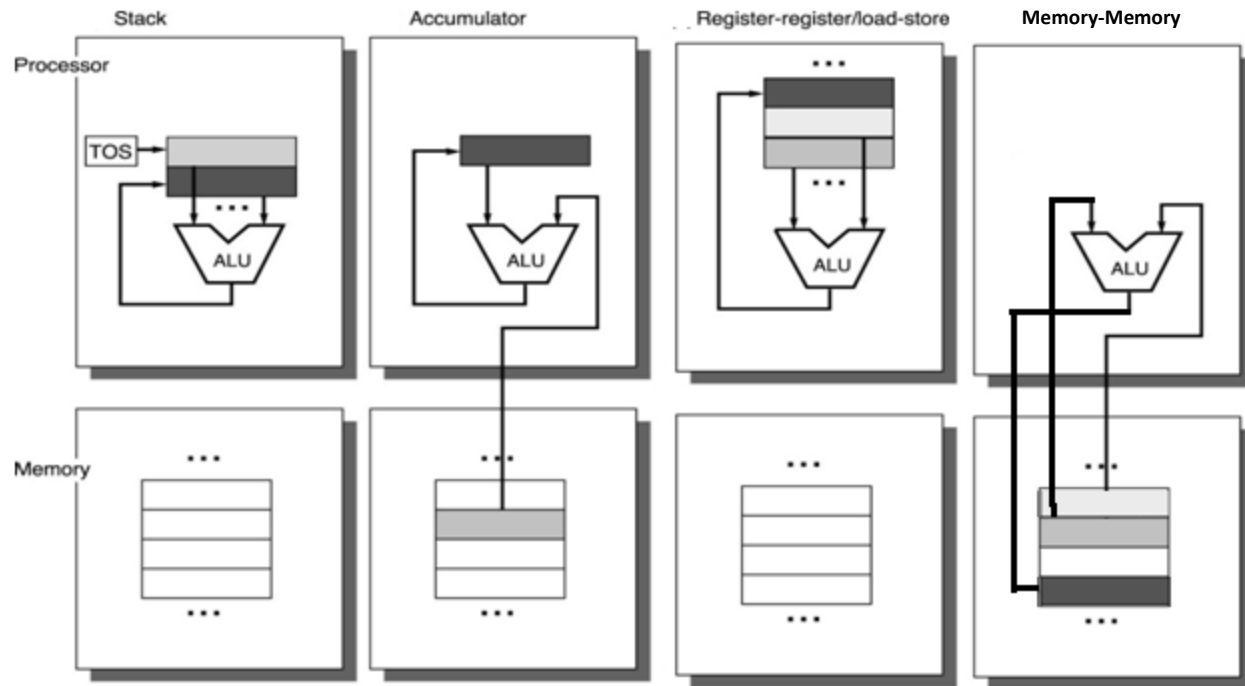
- All operands in memory. Example: DEC VAX.

Source acknowledgement: <http://www.comp.nus.edu.sg/~cs2100/lect/cs2100-13b-ISA.pptx>

Storage Architecture: Example

Stack	Accumulator	Register (load-store)	Memory-Memory
Push A	Load A	Load R1 ,A	Add C , A, B
Push B	Add B	Load R2 ,B	
Add	Store C	Add R3 ,R1,R2	
Pop C		Store R3 ,C	

$$C = A + B$$



Source acknowledgement: <http://www.comp.nus.edu.sg/~cs2100/lect/cs2100-13b-ISA.pptx>

Storage Architecture: **GPR Architecture**

- For modern processors:
 - General-Purpose Register (GPR) is the most common choice for storage design
 - **RISC** computers typically uses **Register-Register (Load/Store)** design
 - E.g. MIPS, ARM, **RISC-V**
 - **CISC** computers use a mixture of Register-Register and Register-Memory
 - E.g. IA32

Source acknowledgement: <http://www.comp.nus.edu.sg/~cs2100/lect/cs2100-13b-ISA.pptx>

Concept #2: Memory & Addressing Mode

- Memory Locations and Addresses
- Addressing Modes

Concept #1: Data Storage

Concept #2: Memory & Addressing Modes

Concept #3: Operations in the Instruction Set

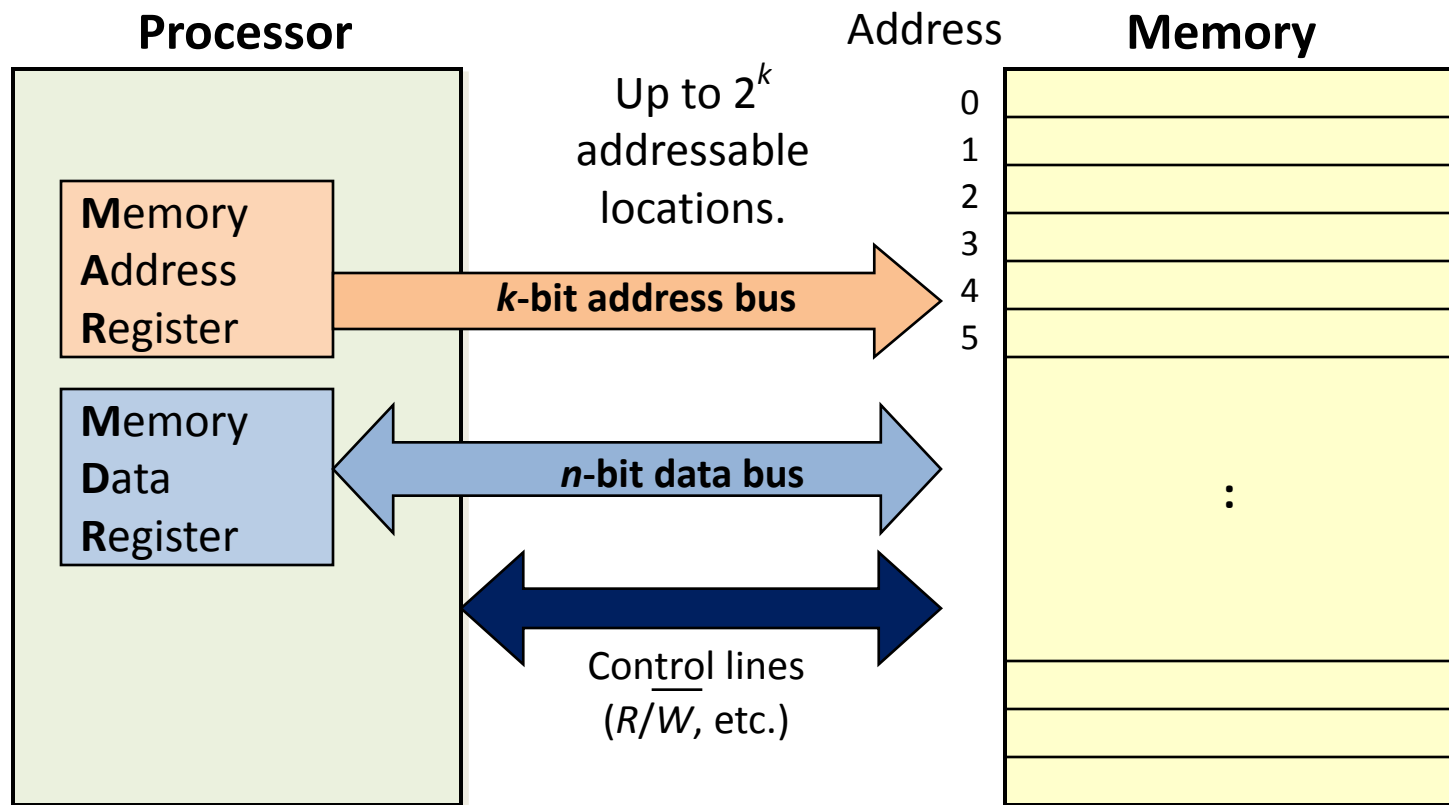
Concept #4: Instruction Formats

Concept #5: Encoding the Instruction Set

Source acknowledgement: <http://www.comp.nus.edu.sg/~cs2100/lect/cs2100-13b-ISA.pptx>

Memory Address and Content

- Given k -bit address, the address space is of size 2^k
- Each memory transfer consists of one word of n bits



Source acknowledgement: <http://www.comp.nus.edu.sg/~cs2100/lect/cs2100-13b-ISA.pptx>

Memory Content: Endianness

■ Endianness:

- The relative ordering of the bytes in a multiple-byte word stored in memory

Big-endian:	Little-endian:																
Most significant byte stored in lowest address. Example: IBM 360/370, Motorola 68000, MIPS (Silicon Graphics), SPARC.	Least significant byte stored in lowest address. Example: Intel 80x86, DEC VAX, DEC Alpha, RISC-V .																
Example: 0xDE AD BE EF Stored as: <table><tr><td>0</td><td>DE</td></tr><tr><td>1</td><td>AD</td></tr><tr><td>2</td><td>BE</td></tr><tr><td>3</td><td>EF</td></tr></table>	0	DE	1	AD	2	BE	3	EF	Example: 0xDE AD BE EF Stored as: <table><tr><td>0</td><td>EF</td></tr><tr><td>1</td><td>BE</td></tr><tr><td>2</td><td>AD</td></tr><tr><td>3</td><td>DE</td></tr></table>	0	EF	1	BE	2	AD	3	DE
0	DE																
1	AD																
2	BE																
3	EF																
0	EF																
1	BE																
2	AD																
3	DE																

Addressing Modes

- Addressing Mode:
 - Ways to specify an operand in an assembly language
- In RISC-V, there are only 4 addressing modes:
 - **Register:**
 - Operand is in a register
 - **Immediate:**
 - Operand is specified in the instruction directly
 - **Displacement:**
 - Operand is in memory with address calculated as $\text{Base} + \text{Offset}$
 - **PC-Relative:**
 - Operand is in memory with address calculated as $\text{PC} + \text{Offset}$

Addressing Modes: Other

<u>Addressing mode</u>	<u>Example</u>	<u>Meaning</u>
Register	Add R4,R3	$R4 \leftarrow R4 + R3$
Immediate	Add R4,#3	$R4 \leftarrow R4 + 3$
Displacement	Add R4,100(R1)	$R4 \leftarrow R4 + \text{Mem}[100 + R1]$
Register indirect	Add R4,(R1)	$R4 \leftarrow R4 + \text{Mem}[R1]$
Indexed / Base	Add R3,(R1+R2)	$R3 \leftarrow R3 + \text{Mem}[R1 + R2]$
Direct or absolute	Add R1,(1001)	$R1 \leftarrow R1 + \text{Mem}[1001]$
Memory indirect	Add R1,@(R3)	$R1 \leftarrow R1 + \text{Mem}[\text{Mem}[R3]]$
Auto-increment	Add R1,(R2)+	$R1 \leftarrow R1 + \text{Mem}[R2]; R2 \leftarrow R2 + d$
Auto-decrement	Add R1,—(R2)	$R2 \leftarrow R2 - d; R1 \leftarrow R1 + \text{Mem}[R2]$
Scaled	Add R1,100(R2)[R3]	$R1 \leftarrow R1 + \text{Mem}[100 + R2 + R3 * d]$

Source acknowledgement: <http://www.comp.nus.edu.sg/~cs2100/lect/cs2100-13b-ISA.pptx>

Concept #3: Operations in Instruction Set

- Standard Operations in an Instruction Set
- Frequently Used Instructions

Concept #1: Data Storage

Concept #2: Memory Addressing Modes

Concept #3: Operations in the Instruction Set

Concept #4: Instruction Formats

Concept #5: Encoding the Instruction Set

Source acknowledgement: <http://www.comp.nus.edu.sg/~cs2100/lect/cs2100-13b-ISA.pptx>

Standard Operations

Data Movement

load (from memory)
store (to memory)
memory-to-memory move
register-to-register move
input (from I/O device)
output (to I/O device)
push, pop (to/from stack)

Arithmetic

integer (binary + decimal) or FP
add, subtract, multiply, divide

Shift

shift left/right, rotate left/right

Logical

not, and, or, set, clear

Control flow

Jump (unconditional), Branch (conditional)

Subroutine Linkage

call, return

Interrupt

trap, return

Synchronization

test & set (atomic r-m-w)

String

search, move, compare

Graphics

pixel and vertex operations,
compression/decompression

Source acknowledgement: <http://www.comp.nus.edu.sg/~cs2100/lect/cs2100-13b-ISA.pptx>

Frequently Used Instructions

Rank	Integer Instructions	Average Percentage
1	Load	22%
2	Conditional Branch	20%
3	Compare	16%
4	Store	12%
5	Add	8%
6	Bitwise AND	6%
7	Sub	5%
8	Move register to register	4%
9	Procedure call	1%
10	Return	1%
Total		96%

Make these instructions fast!
Amdahl's law – make the
common case fast!

Source acknowledgement: <http://www.comp.nus.edu.sg/~cs2100/lect/cs2100-13b-ISA.pptx>

Concept #4: Instruction Formats

- Instruction Length
- Instruction Fields
 - Type and Size of Operands

Concept #1: Data Storage

Concept #2: Memory Addressing Modes

Concept #3: Operations in the Instruction Set

Concept #4: Instruction Formats

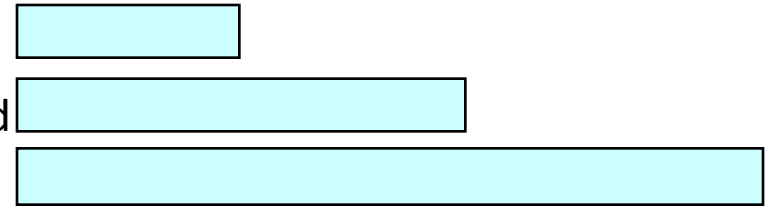
Concept #5: Encoding the Instruction Set

Source acknowledgement: <http://www.comp.nus.edu.sg/~cs2100/lect/cs2100-13b-ISA.pptx>

Instruction Length

- **Variable-length** instructions.

- Intel 80x86: Instructions vary from 1 to 17 bytes long.
- Digital VAX: Instructions vary from 1 to 54 bytes long.
- Require multi-step fetch and decode.
- Allow for a more flexible (but complex) and compact instruction set.



- **Fixed-length** instructions.

- Used in most RISC (Reduced Instruction Set Computers)
- MIPS, PowerPC: Instructions are 4 bytes long.
- Allow for easy fetch and decode.
- Simplify pipelining and parallelism.
- Instruction bits are scarce.

- **Hybrid** instructions: a mix of variable- and fixed-length instructions.

Source acknowledgement: <http://www.comp.nus.edu.sg/~cs2100/lect/cs2100-13b-ISA.pptx>

Instruction Fields

- An instruction consists of
 - **opcode** : unique code to specify the desired operation
 - **operands**: zero or more additional information needed for the operation
- The operation designates the type and size of the operands
 - **Typical type and size**: Character (8 bits), half-word (eg: 16 bits), word (eg: 32 bits), single-precision floating point (eg: 1 word), double-precision floating point (eg: 2 words).
- Expectations from any new 32-bit architecture:
 - Support for 8-, 16- and 32-bit integer and 32-bit and 64-bit floating point operations. A 64-bit architecture would need to support 64-bit integers as well.

Source acknowledgement: <http://www.comp.nus.edu.sg/~cs2100/lect/cs2100-13b-ISA.pptx>

Concept #5: Encoding the Instruction Set

- Instruction Encoding
- Encoding for Fixed-Length Instructions

Concept #1: Data Storage

Concept #2: Memory Addressing Modes

Concept #3: Operations in the Instruction Set

Concept #4: Instruction Formats

Concept #5: Encoding the Instruction Set

Source acknowledgement: <http://www.comp.nus.edu.sg/~cs2100/lect/cs2100-13b-ISA.pptx>

Instruction Encoding: Overview

- How are instructions represented in binary format for execution by the processor?
- Issues:
 - Code size, speed/performance, design complexity.
- Things to be decided:
 - Number of registers
 - Number of addressing modes
 - Number of operands in an instruction
- The different competing forces:
 - Have many registers and addressing modes
 - Reduce code size
 - Have instruction length that is easy to handle (fixed-length instructions are easy to handle)

Source acknowledgement: <http://www.comp.nus.edu.sg/~cs2100/lect/cs2100-13b-ISA.pptx>

Encoding Choices

- Three encoding choices: variable, fixed, hybrid.

Operation and no. of operands	Address specifier 1	Address field 1	...	Address specifier	Address field
----------------------------------	------------------------	--------------------	-----	----------------------	------------------

(a) Variable (e.g., VAX, Intel 80x86)

Operation	Address field 1	Address field 2	Address field 3
-----------	--------------------	--------------------	--------------------

(b) Fixed (e.g., Alpha, ARM, MIPS, PowerPC, SPARC, SuperH)

Operation	Address specifier	Address field
-----------	----------------------	------------------

Operation	Address specifier 1	Address specifier 2	Address field
-----------	------------------------	------------------------	------------------

Operation	Address specifier	Address field 1	Address field 2
-----------	----------------------	--------------------	--------------------

(c) Hybrid (e.g., IBM 360/70, MIPS16, Thumb, TI TMS320C54x)

Source acknowledgement: <http://www.comp.nus.edu.sg/~cs2100/lect/cs2100-13b-ISA.pptx>

Fixed Length Instruction: Encoding (1/4)

- Fixed length instruction presents a much more interesting challenge:
 - Q: How to fit multiple sets of instruction types into same number of bits?
 - A: Work with the most constrained instruction types first
- **Expanding Opcode** scheme:
 - The opcode has variable lengths for different instructions.
 - A good way to maximizes the instruction bits.

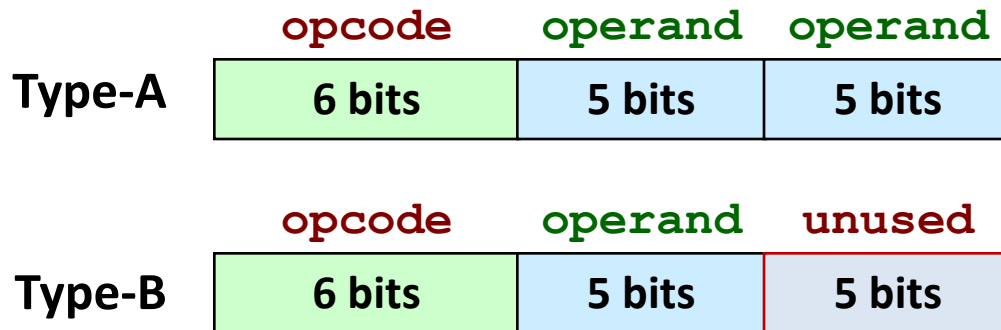
Source acknowledgement: <http://www.comp.nus.edu.sg/~cs2100/lect/cs2100-13b-ISA.pptx>

Fixed Length Instruction: Encoding (2/4)

- **Example:**

- 16-bit fixed length instructions, with 2 types of instructions
- **Type-A:** 2 operands, each operand is 5-bit
- **Type-B:** 1 operand of 5-bit

First Attempt:
Fixed length Opcode



Problem:

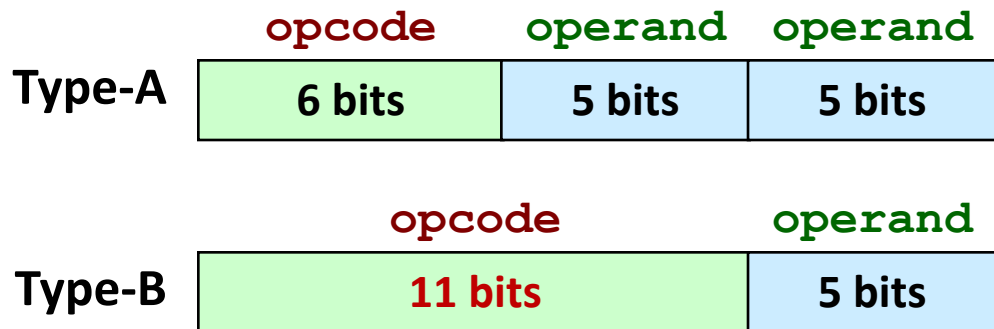
- Wasted bits in Type-B instruction
- Maximum total number of instructions is 2^6 or 64.

Source acknowledgement: <http://www.comp.nus.edu.sg/~cs2100/lect/cs2100-13b-ISA.pptx>

Fixed Length Instruction: Encoding (3/4)

- Use **expanding opcode** scheme:
 - Extend the opcode for type-B instructions to **11 bits**
 - ➔ No wasted bits and result in a larger instruction set

Second Attempt:
Expanding Opcode



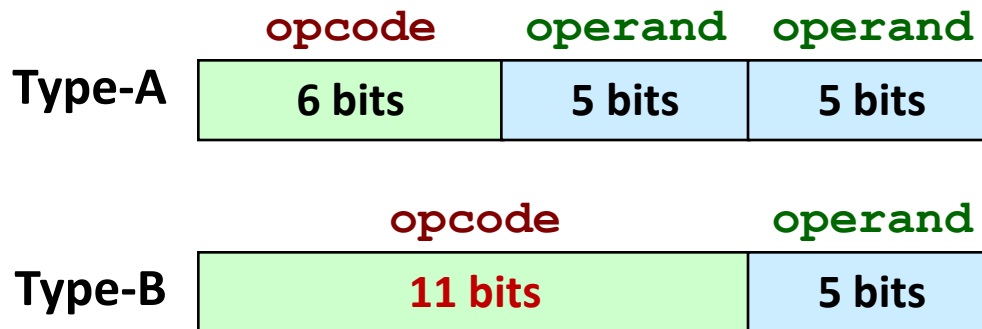
■ Questions:

- How do we distinguish between Type-A and Type-B?
- How many different instructions do we really have?

Source acknowledgement: <http://www.comp.nus.edu.sg/~cs2100/lect/cs2100-13b-ISA.pptx>

Fixed Length Instruction: Encoding (4/4)

- What is the maximum number of instructions?



Answer:

$$\begin{aligned} & 1 + (2^6 - 1) \times 2^5 \\ &= 1 + 63 \times 32 \\ &= 2017 \end{aligned}$$

■ Reasoning:

- For every 6-bit prefix (front-part) given to Type-B, we get 2^5 unique patterns, e.g. [111111]XXXXXX
- So, we should minimize Type-A instruction and give as many 6-bit prefixes as possible to Type-B

➔ 1 Type-A instruction, $2^6 - 1$ prefixes for Type-B

Source acknowledgement: <http://www.comp.nus.edu.sg/~cs2100/lect/cs2100-13b-ISA.pptx>