CZ1006
CE1006

Instruction Organisation in Memory

A/P Goh Wooi Boon

Instruction Organisation in Memory

Characteristics and Role of Instructions

Learning Objectives (2.2a)

1. Describe the characteristics and role of an instruction.
2. Describe the VIP programmer's model.
3. Describe the function and interpretation of various VIP registers.

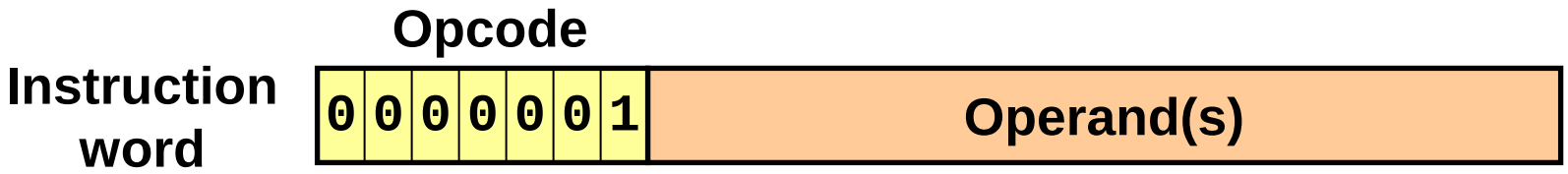
Instruction Organization in Memory

- Instructions are binary encoded and stored in memory. Unlike data, instruction bytes tell CPU what actions to take (i.e. they are **executable**).
- Most instruction formats consist of **two** parts.

  - First part of instruction (**op-code**) specifies the operation to be carried out (e.g. add, subtract).
  - Remaining bits (**operands**) specify the data itself or the location of data and where results is to be stored.
  - Some operations produce no storable result but may alter program sequence or influence status flags (e.g. V,N,Z,C).

Opcode and Operands

- **How are opcode encoded in an instruction?**
  - If there are 80 different operations, (e.g. **add**, **sub**, etc) then at least 7 bits ($2^6 < 80 < 2^7$) of opcode is needed to represent all the unique bit patterns.



- **How many ways are there to specify the operand(s) to be operated on**?
  - Numerous. An operand can be stored as part of the instruction, stored in a register or in a memory location.
  - The method by which the operand is specified is known as **addressing mode**.

Addressing Mode Examples

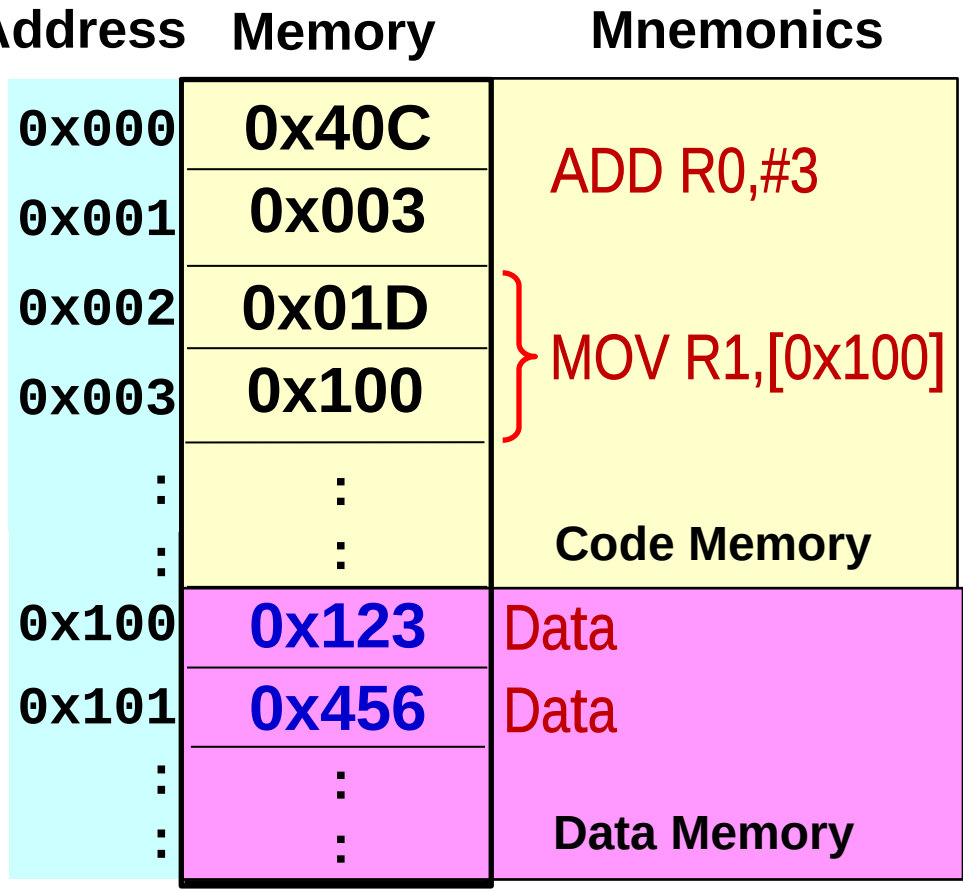
Addressing Mode	VIP	ARM
Absolute	`MOV R0,[0x100]`	None
Register Direct	`ADD R0,R1`	`ADD r0,r1,r2`
Immediate	`ADD R0,#3`	`ADD r3,r3,#3`
Register Indirect	`MOV R0,[R1]`	`LDR r0,[r1]`
Register Indirect with Offset	`MOV R0,[R2+4]`	`LDR r0,[r1,#4]`
Register Indirect with Index	None	`LDR r0,[r1,r2]`
Implied	`JNE -8`	`BNE LOOP`

Instructions and Data in Memory

- Instructions are stored in memory as binary patterns called **machine codes**.
- They are also represented in more readable **mnemonics**.
  - VIP (12-bit processor) example:

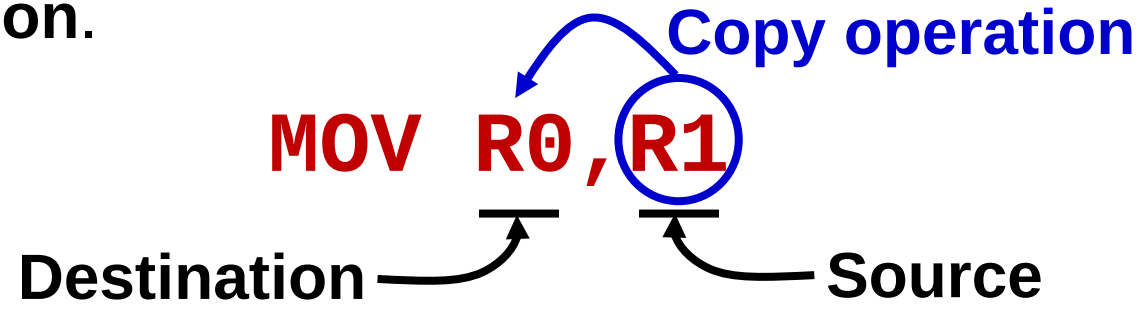
```
ADD R0,#3
MOV R1,[0x100]
:
```

  - Memory is partitioned into separate **code** and **data** segments.



VIP Instruction Format

- Introduction to two simple VIP mnemonics.
  - The **MOV** operator is a two-operand instruction that copies the source operand to the destination operand.
  - The right operand is the **source** and left operand is the **destination**.



  - The **ADD** operator is a two-operand instruction that adds the source and destination operands and put the result in the **destination** operand.

Role of Instructions

- The role of an instruction is to make purposeful **changes** to the **current state** of the processor**.**
  - The visible current state of a processor is defined by the **programmer's model** and contents in **memory**.
  - There are three broad categories of instructions for general programming available in a processor:

Data Processing

VIP examples:

```
ADD  R0,#3
OR   R1,R2
ROR  R3
```

Data Transfer

VIP examples:

```
MOV  R0,#3
MOV  R0,[0x100]
POP  R1
```

Program Control

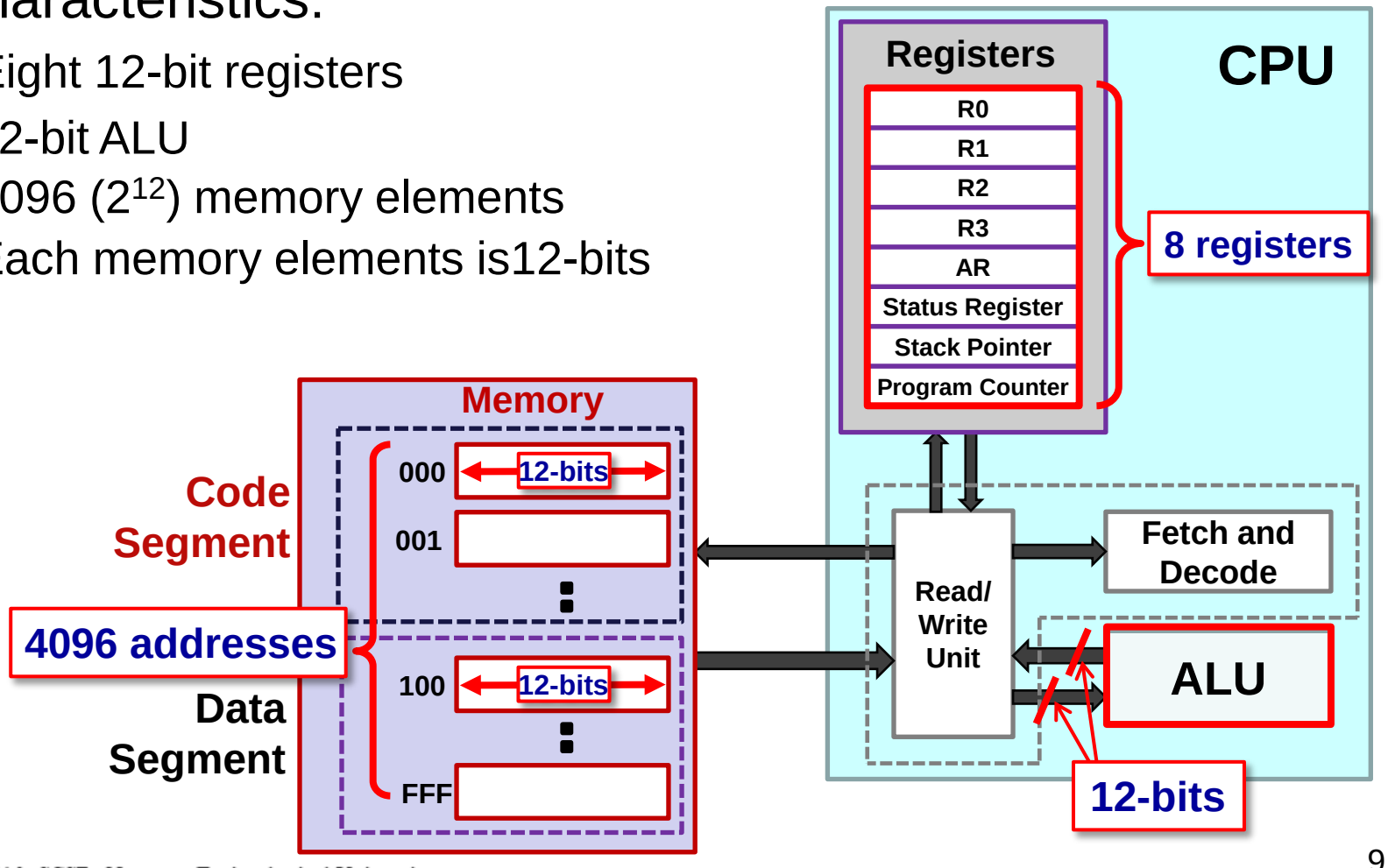
VIP examples:

```
JMP  3
JGE  0xFFC
CALL Routine
```

  - Most processors also have instructions that affect system-level operation in the CPU (e.g. **HALT**, **BCSR**)

The Various Instruction Paradigms (VIP) Processor

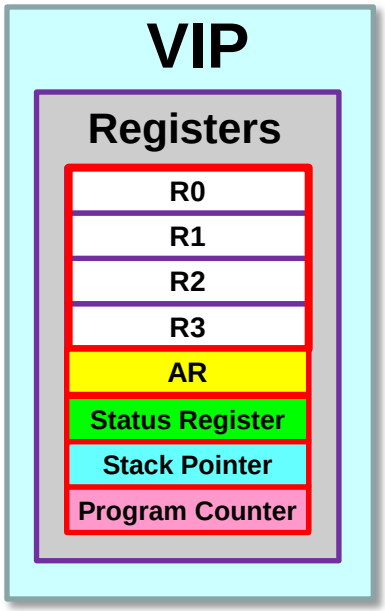
- VIP is a 12-bit simulated processor designed for teaching multiple principles in Instruction Set Architectures (ISA)
- Characteristics:
  - Eight 12-bit registers
  - 12-bit ALU
  - 4096 (2^{12}) memory elements
  - Each memory elements is12-bits



VIP Programmer's functional model

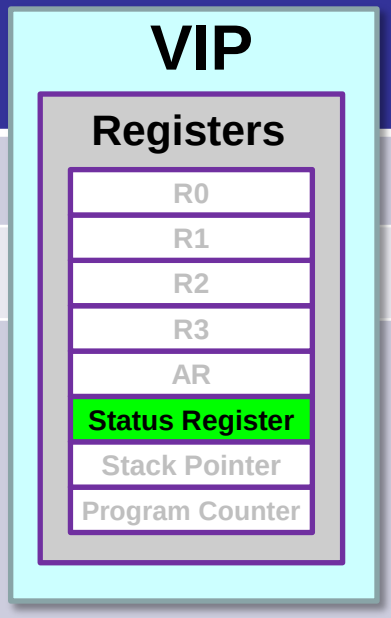
- To the programmer, VIP has eight 12-bit registers for storing data, addresses of data and status information in order to co-ordinate program execution.

- General purpose registers:
  - **R0, R1, R2, R3** can be used to store data or addresses.
- Auxiliary register:
  - **AR** may be used to store data (in some instructions, the use of AR is implicit).
- Specialised registers:
  - **SR** contains flags that are set according to outcome of different instructions or events.
  - **SP** is used to manage the system stack.
  - **PC** is use by the control unit to sequence the execution of the program and is set to 0 when processor is reset.



VIP's status register and flags

SR bit(s)	Name	State at Reset	Description
11-8	*	*	Reserved
7-4	*	*	Defined but not described here
3	**V**	0	Set if sign of result is incorrect using 2's complement For addition: if msb(s)=msb(d) and msb(result)≠msb(d) For subtraction: if msb(s)≠msb(d) and msb(result)≠msb(d)
2	**N**	0	Corresponds to most significant bit of the result
1	**Z**	0	Set if result of an operation was zero, otherwise cleared
0	**C**	0	Set if carry out, otherwise cleared



V – Overflow; **N** – Negative; **Z** – Zero; **C** – Carry

Program Counter

- A special register in the CPU called the Program Counter (**PC**) is used to keep track of **program execution.**
  - The content of the PC is an **address**.
  - This address is where the **next instruction** to be executed is stored.
  - The PC **automatically increments** by the length of the instruction that has been executed.
  - However, this sequential execution can be altered by modifying the contents of the PC to **any new address** location (i.e. a jump or a branch operation)
  - The PC is set to address **0x000** when processor is first powered up or goes into **reset**.

Summary

- An instruction usually consist of a **opcode** and an **operand**.
- Instructions in a program changes the **states** of a processor as it executes.
- The states of the processor consist of values in:
  - general purpose registers (e.g. data & address registers)
  - specialised function registers (e.g. SR and PC)
  - memory locations
- The design of a program is to ensure that these **states are changed** in a purposeful manner during the execution of the sequence of instructions.

Instruction Organisation in Memory

Basic Execution Cycle

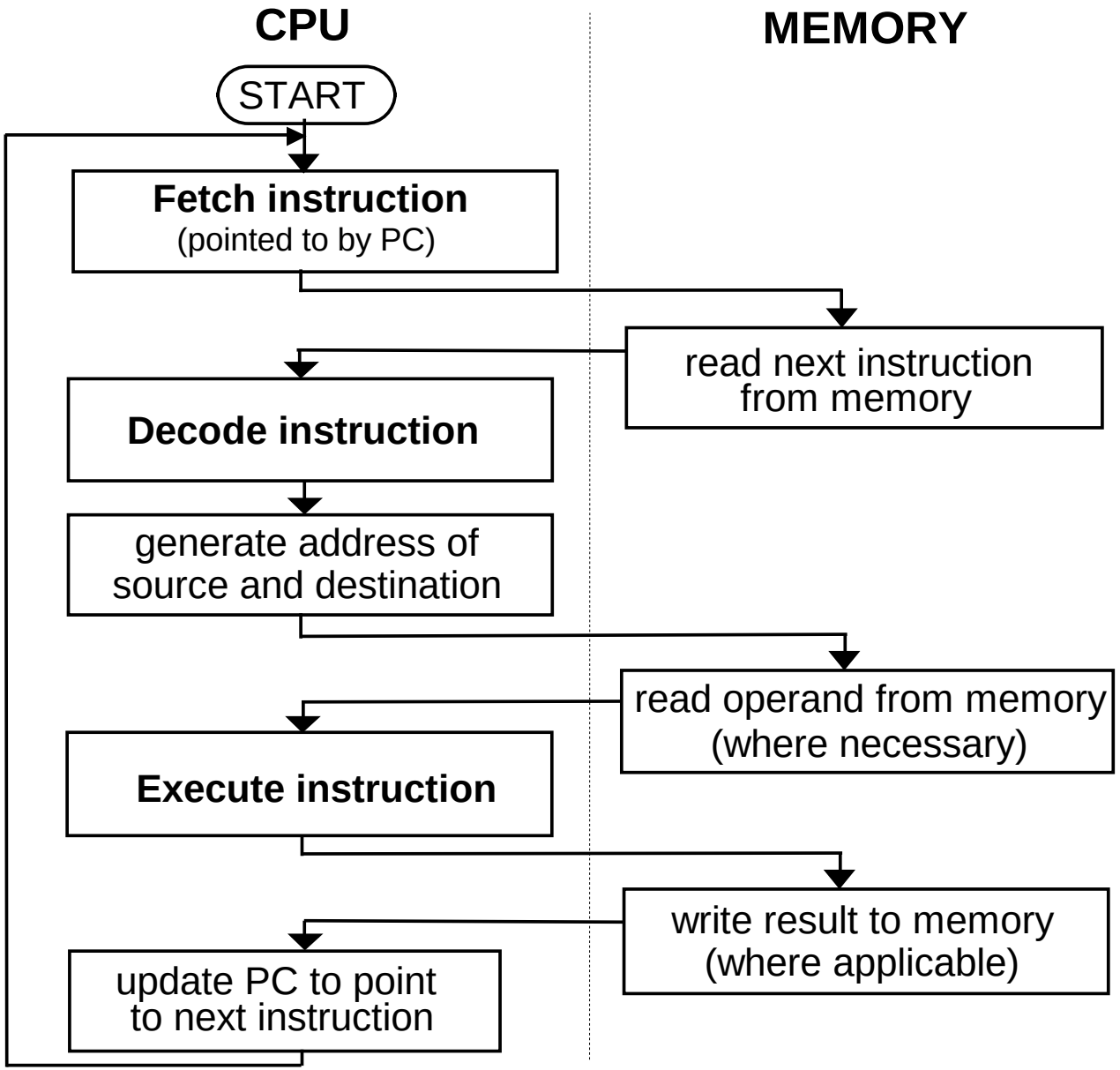
Learning Objectives (2.2b)

1. Describe the basic execution cycle for a simple ADD instruction.

Basic Execution Cycle

- What are the steps in executing an instruction?
  - **Fetch instructions** - CPU read instructions from memory.
  - **Decode instructions** - Instruction is decoded in order to determine action required.
  - **Fetch data** – Some data from memory may need to be fetched in order to complete execution of instruction.
  - **Execute** – Instruction execution may require CPU to perform some arithmetic or logical operation on the data.
  - **Write data** – Result of execution may need to be stored in CPU temporarily or to stored back to memory.
  - This **fetch-decode-execute** cycle is performed repeatedly by the CPU once powered-on.

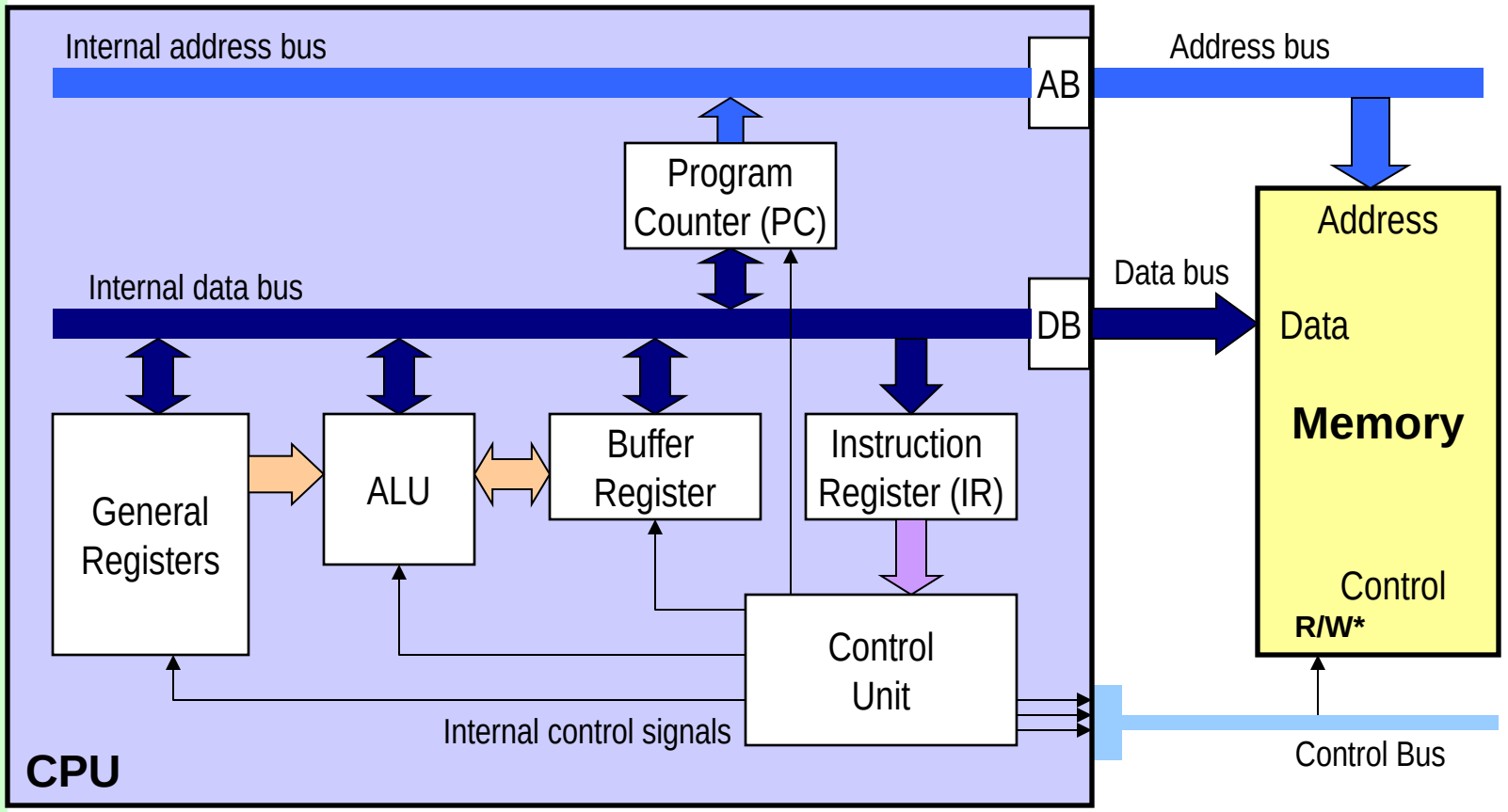
Typical Instruction Execution Flowchart



Basic Execution Cycle

A Simple Processor

- Basic components of a simple processor (CPU) and its interface signals to external main memory.

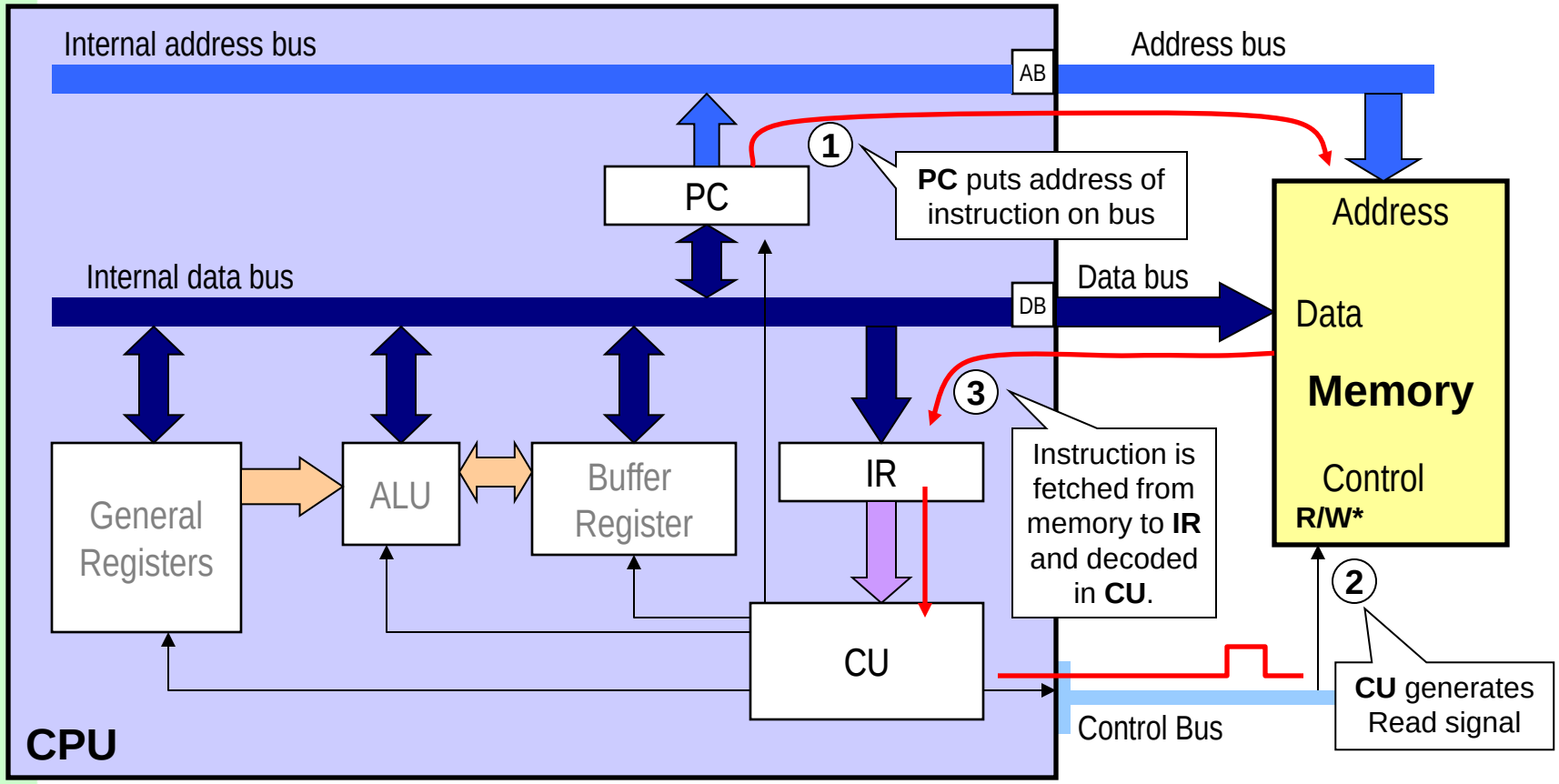


AB = Address buffer/latch
DB = Data buffer

Basic Execution Cycle

Fetch Cycle - Instruction

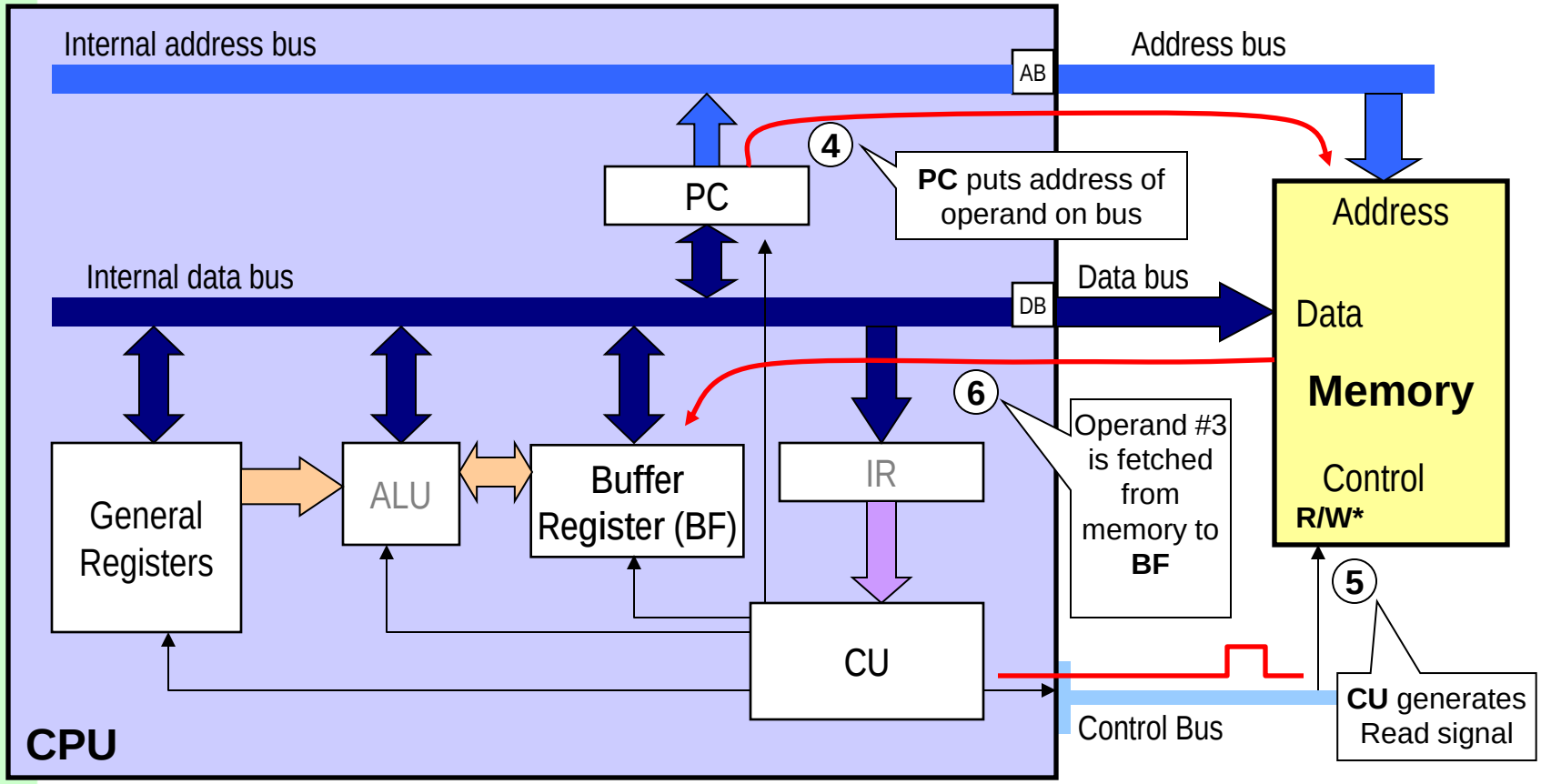
- Consider an instruction like: **`ADD R0,#3.`** In the fetch cycle, the **PC** points to address of next the instruction and its opcode is fetched into the **IR**.



Basic Execution Cycle

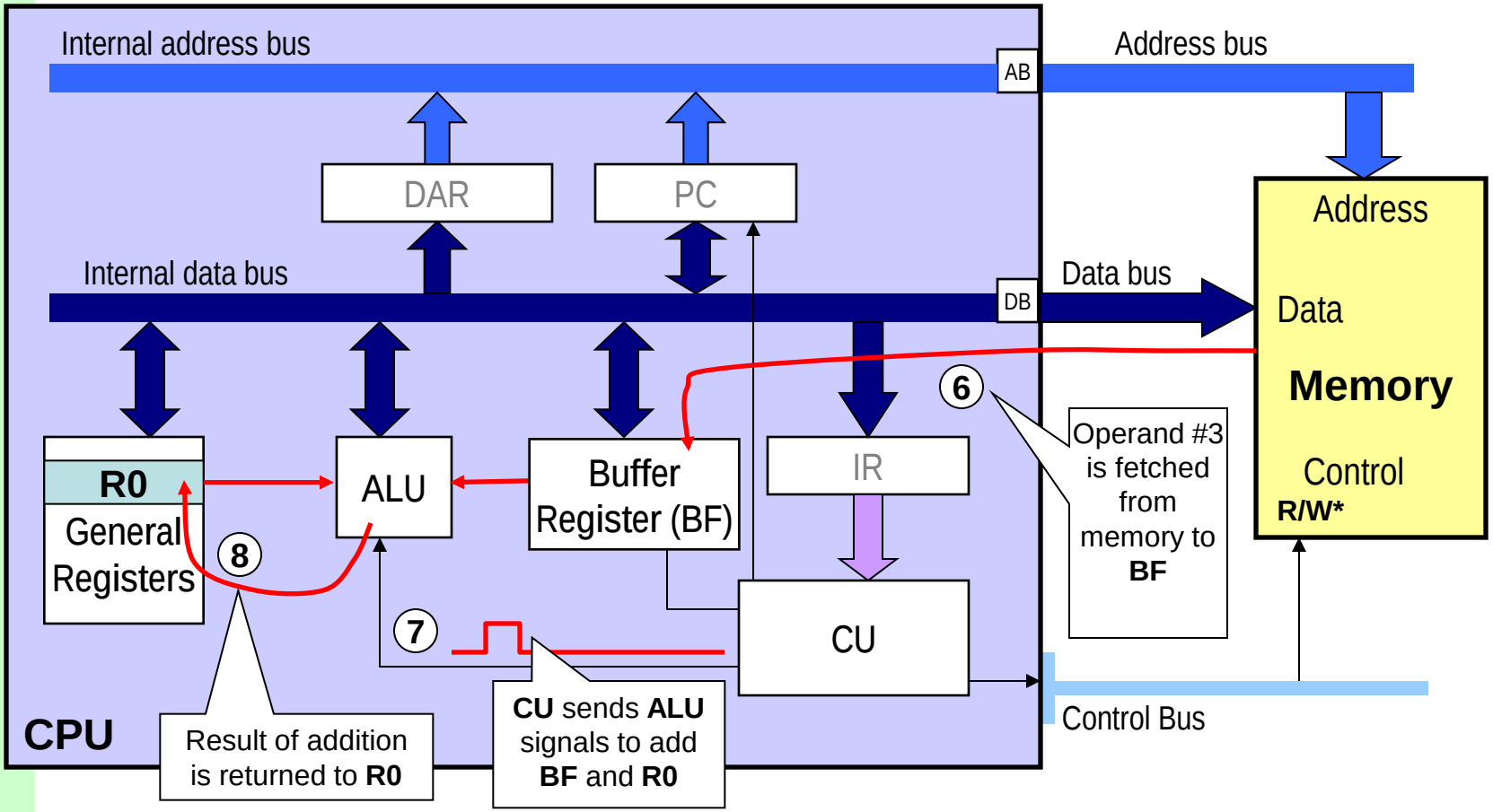
Fetch Cycle - Operand

- Decoding **`ADD R0,#3`** suggest an operand is needed. Since its is stored in memory, another fetch is required to retrieve operand into CPU.



Execute Cycle - Add

- In the execution cycle, the operand **#3** and the contents in register **R0** are added and the result is stored back into **R0**, replacing its original content.



Summary

- Execution of a single instruction may involve multiple accesses to memory. In most single memory (von Neumann-type) CPU, different instructions take **different number** of clock **cycles** to execute.
- Data transfer on external bus is slower than within CPU's internal bus. μP system performance is limited by data traffic bandwidth between CPU and memory (also known as the **von Neumann bottleneck**).
- Keeping regularly used operands in CPU registers helps reduce memory access.
- Keeping instructions and data in separate memories (**Harvard architecture**) can help make instructions execute in more regular cycles (using parallel fetches) and thus improve performance.

Instruction Organisation in Memory

Instruction Encoding

Learning Objectives (2.2c)

1. Describe how a typical VIP instruction is encoded.
2. Describe the differences between variable & fixed-length Instruction Set Architectures (ISA).
3. Describe characteristics of the Load-Store architecture.

Instruction Encoding

- For CPU to identify each unique assembly instruction, they must be encoded with unique binary patterns.
  - 12-bit machine like VIP will encode instructions in basic units of 12 bits.
- Both the op-code and the operands must be encoded.
  - Instructions like **`MOV`** and **`ADD`** must encode two operands.

- What are some of the factors influencing the manner instructions are encoded?
  - Number of operands supported, e.g. VIP (2), ARM (3).
  - Number of registers in CPU, e.g. VIP (8), ARM (16).
  - Number of different addressing modes supported.

Operand Encoding in VIP

11	10	9	8	7	6	5	4	3	2	1	0
0-7	Dual operand			d				s			
8	Short Move			d				n			
9-A	Unary or Control			Operation				s/d or n			
B-F	JMP			2's complement -128 to +127 relative displacement							

Hex	s, d	Source or destination data
0	R0	Register R0
1	R1	Register R1
2	R2	Register R2
3	R3	Register R3
4	[R0]	Register R0 indirect
5	[R1]	Register R1 indirect
6	[R2+n]	Register R2 plus offset indirect
7	[R3+n]	Register R3 plus offset indirect
8	AR	Auxiliary Register
9	SR	Status Register
A	SP	Stack Pointer
B	PC	Program Counter
C	#n	Immediate
D	n or [n] or &n	Absolute
E	[SP+n]	Register SP plus offset indirect
F	[PC+n]	Register PC plus offset indirect

- Encoding for **`ADD R2,R1`** is

  `0100 0010 0001` (421)

  12 bits (1 word)

- What happens when one of the operand is 12 bits?

  e.g. **`ADD R0,#3`**

- The encoding must be in two 12-bit values:

  2 words

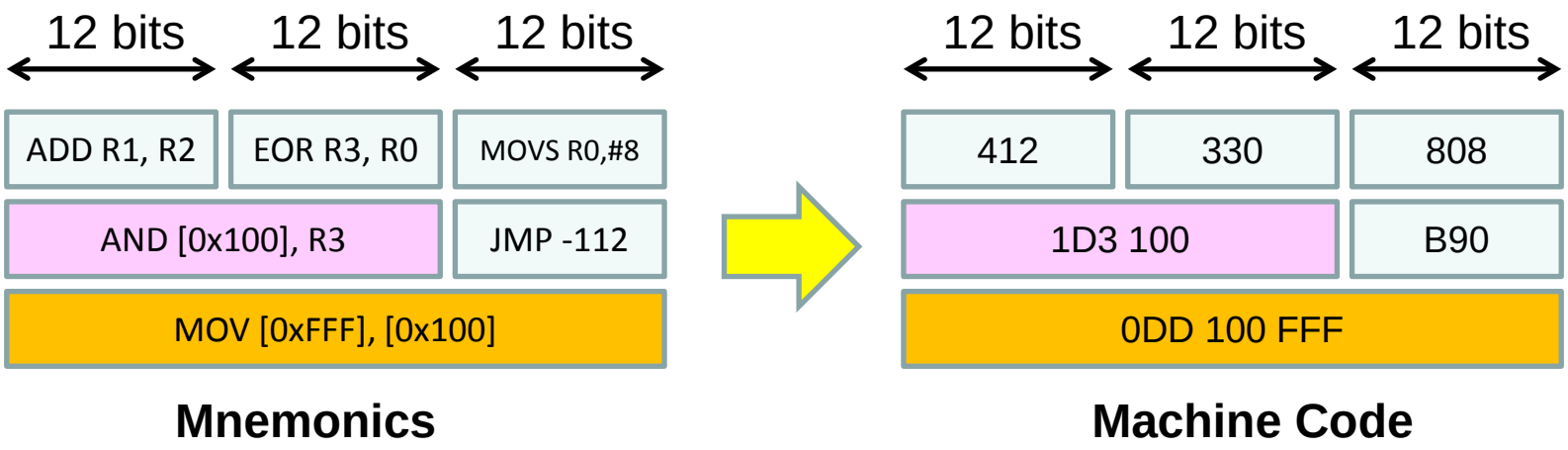
  `0100 0000 1100` (40C)

  `0000 0000 0011` (003)

Variable-length Instruction Encoding

- VIP requires variable-length instructions to handle operands that cannot be encoded in a single 12-bit instruction.

VIP



ADD R1, R2

- Register-to-register operations can be encoded in a single word.

AND [0x100], R3

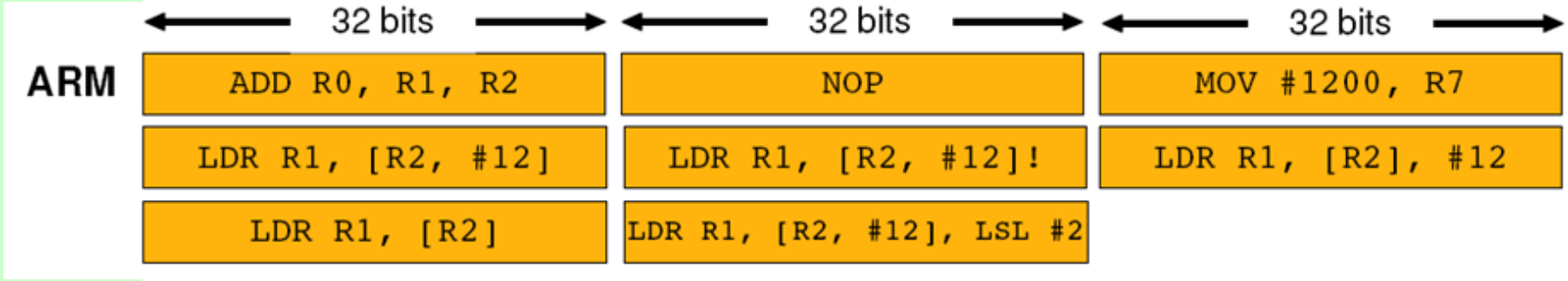
- Register-to-memory operations need an additional word to encode the destination memory address.

MOV [0xFFF], [0x100]

- Memory-to-memory operations need two more words to encode both source & destination memory addresses.

Fixed-length Instruction Encoding

- Unlike the VIP, the 32-bit ARM ISA uses fixed-length 32-bit instructions.

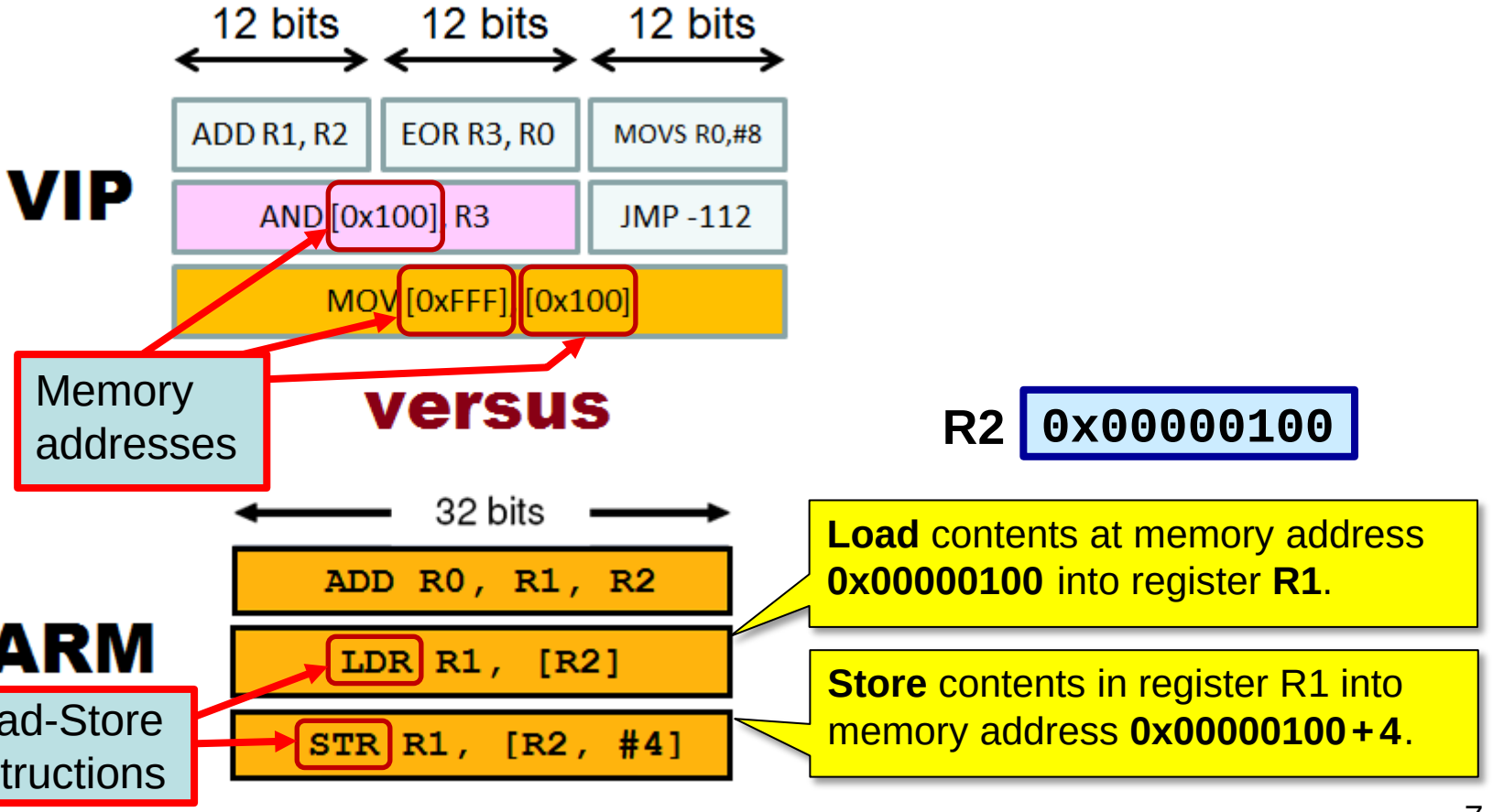


- With fixed-length instruction encoding, both opcode and operands must be specified within **one** instruction word.
- This can be achieved by restricting most operations to involve **only registers**.
- To avoid extension words, an immediate value in the 32-bit ARM ISA is specified with only **12 bits**. As such, not every combinations in the 32-bit value can be specified.

Google "***Specifying immediate value in ARM ISA***"

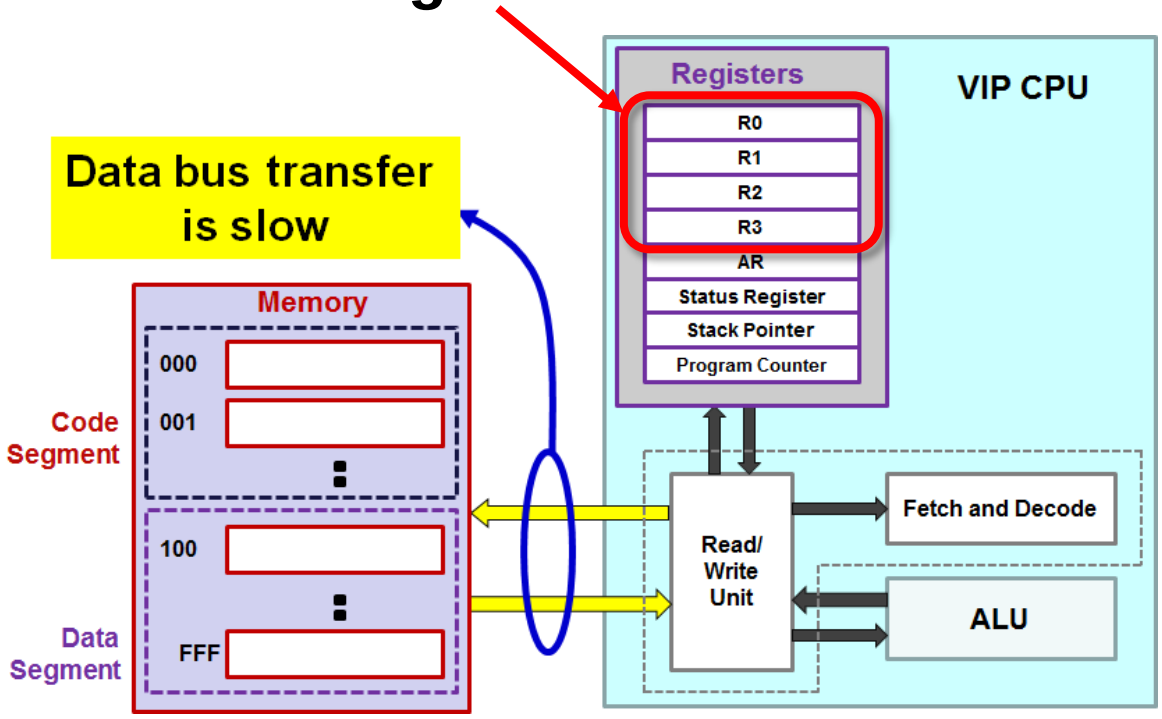
Load-Store Architecture

- To achieve fix-length instructions, restrictions must be placed on encoding **operand addresses** in instructions.
- Load-Store architectures (e.g. in ARM) access memory contents via values that are already **in** the **registers**.



Using Registers

- Operations using registers are **faster** than those involving memory.
  - Data transfer between registers and ALU is faster due to it physically proximity.
  - To speed up code execution, keep as much of your computations within **registers**.



Implication of Register Count

- Implementing many registers within the CPU will incurs increasing overheads.
  - They take up precious **space** in the silicon die.
  - Connections to various busses increase the **routing** and **multiplexing complexity**.
  - More registers increases the **operand size** during instruction encoding.

ADD (3)

15	14	13	12	11	10	9	8 – 6	5 – 3	2 – 0
0	0	0	1	1	0	0	Rm	Rn	Rd

This form of ADD adds the value of one register to the value of a second register, and stores the result in a third register. The condition code flags are updated, based on the result.

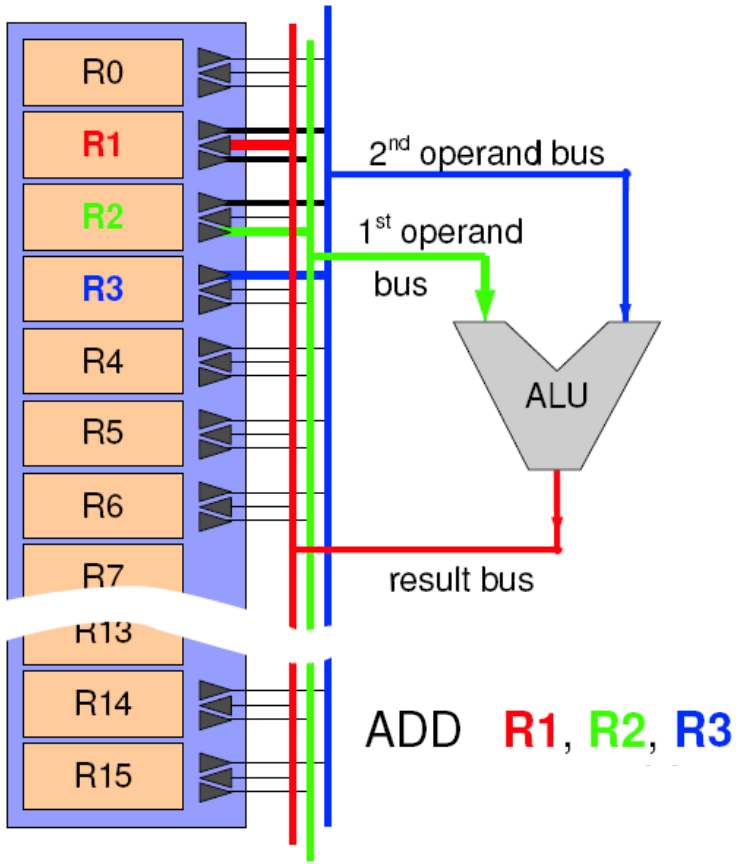
Syntax

`ADD <Rd>, <Rn>, <Rm>`

where:

`<Rd>`	Is the destination register for the completed operation.
`<Rn>`	Specifies the register containing the first value for the addition.
`<Rm>`	Specifies the register containing the second value for the addition.

Instruction Encoding for ADD in Thumb-2



Orthogonality of ISA

- In a truly orthogonal ISA, every instruction is able to use every available addressing modes.
  - A truly orthogonal ISA does not restrict certain instruction from using only specific registers.
- Difficult to achieve complete orthogonality due to the **large number of bit patterns** required to express all combinations of operations and addressing modes.
  - Increase instruction length will increases the memory size required by the program.

Summary

- The VIP ISA uses **variable-length** encoding.
  - Using only registers can be keep instructions to 1 word.
  - Operands consisting of immediate values and memory addresses required extension words to encode them.
- The ARM 32-bit ISA uses **fixed-length** encoding.
  - The Load-Store architecture accesses memory through the use of address values that are already in registers.
- Maximise the use of **register-based operations** to optimise you code for speedy execution.