PES UNIVERSITY ONLINE

RISC V Architecture

Mahesh Awati
Department of Electronics and Communication Engg.

RISC V ARCHITECTURE

UNIT 2 – Instructions: The Language of Computer

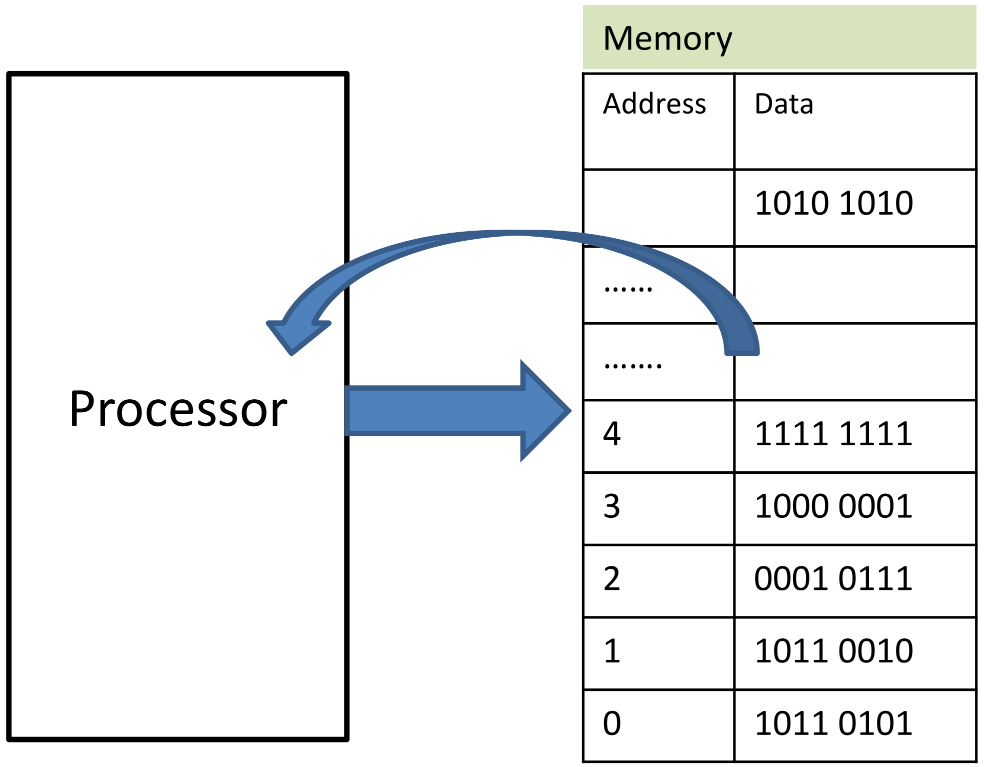
Mahesh Awati
Department of Electronics and Communication Engineering

Instructions – Language of Computer

Operands of the Computer Hardware

2. Memory Operands

- If Programming languages have more complex data **structures i.e., arrays and structures.**
- The number of elements in these data structure **may be more than the registers available** in a computer hardware (to much to fit in 32 registers).
- **Memory is Large** but **slow**
- Memory is just a large, 1D array, with the address acting as the index to that array, starting at 0.
- Memory locations are addressed to access the data.



Memory

Address	Data
	1010 1010
……	
…….	
4	1111 1111
3	1000 0001
2	0001 0111
1	1011 0010
0	1011 0101

Operands of the Computer Hardware

How the memory is addressed ?

Word Addressable: What is Word addressable Memory?

- Each 32-bit data word has a unique address
- Using the unique address , a 32 bit data can be read/written and Next 32 bit data will be there is next successive location

Word No	Word Address	Data (32 bits = 4bytes) 31 24	23 16	15 8	7 0
10	0x000A	AB	FD	12	34
	………..				
2	0x0002	12	03	DF	2D
1	0x0001	DD	FF	F2	2A
0	0x0000	12	AA	BB	B9h 1011 1001b

Instructions – Language of Computer

Operands of the Computer Hardware

How the memory is addressed ?

Byte Addressable: What is byte addressable Memory?

- Each data byte has a unique address (Individual bytes have individual address)
- A 32-bit word = 4 bytes, so **word address increments by 4**

Word No	Word Address	Data (32 bits = 4bytes)			
		31 24	23 16	15 8	7 0
4	0x0010	AB	FD	12	34
3	0x000C				
2	0x0008	12	03	DF	2D
1	0x0004	DD	FF	F2	2A
0	0x0000	12	AA	BB	B9h

Order of Data Storage:

Little Endian: Base address holding byte0 of a word

Ex: Address 0x0004 = 0xDDFFF22A

Big Endian: Base address holding byte3 of a word

Ex: Address 0x0004 = 0x2AF2FFDD

Byte Address

0X0013	0X0012	0X0011	0X0010
0X000F	0X000E	0X000D	0X000C
0X000B	0X000A	0X0009	0X0008
0X0007	0X0006	0X0005	0X0004
0X0003	0X0002	0X0001	0X0000

Base Address of words

Reference : Computer Architecture with RISC V - The Hardware/Software Interface: RISC-V Edition by David A. Patterson and John L. Hennessy

Instructions – Language of Computer

Operands of the Computer Hardware

How the memory is addressed ?

Byte Addressable: What is byte addressable Memory?

- Each data byte has a unique address (Individual bytes have individual address)
- A 64-bit double-word = 8 bytes, so **double-word address increments by 8**

D-Word No	Word Address	Data (32 bits = 4bytes) 64						15 8	7 0
4	0x0020	AB	45	FD	ED	12	AA	34	13
3	0x0018								
2	0x0010	12	DA	03	55	DF	76	2D	AA
1	0x0008	DD	44	FF	AE	F2	0F	2A	11
0	0x0000	12	11	AA	A1	BB	E2	B9h	DE

Byte Address

0X27	0X26	0X25	0X24	0X23	0X22	0X21	0X20
0X1F	0X1E	0X1D	0X1C	0X1B	0X1A	0X19	0X18
0X17	0X16	0X15	0X14	0X13	0X12	0X11	0X10
0X0F	0X0E	0X0D	0X0C	0X0B	0X0A	0X09	0X08
0X07	0X06	0X05	0X04	0X03	0X02	0X01	0X00

Instructions – Language of Computer

Operands of the Computer Hardware

2. Memory Operands

How a computer represent and access such complex data **structures?**

- The **complex data structure (array and structure)** is not stored in registers , instead the compiler **allocates complex data structure to locations in memory**.
- The compiler then place the **starting address into the data transfer instructions**.
- Let us consider array A of 100 double-words stored in memory by a compiler. The starting address 0x10 is referred as base address in which A[0] is stored.

Starting Address of Array which is referred as base address

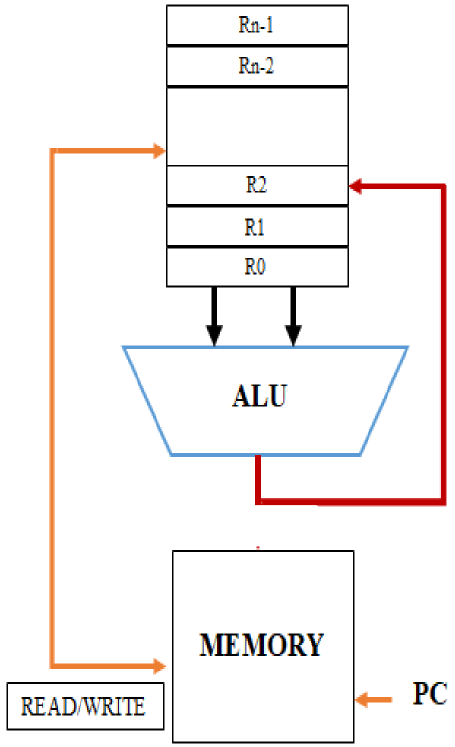
Element	Address	Data
99	0x330	A[99]
….		A[……]
8	0x58	A[8]
….	…….	Data (64 bit)
2	0x20	A[2]
1	0x18	A[1]
0	0x10	A[0]
	0x08	Data (64 bit)
0	0x00	Data (64 bit)

Operands of the Computer Hardware

2. Memory Operands

How RISC V represent and access such large structures?

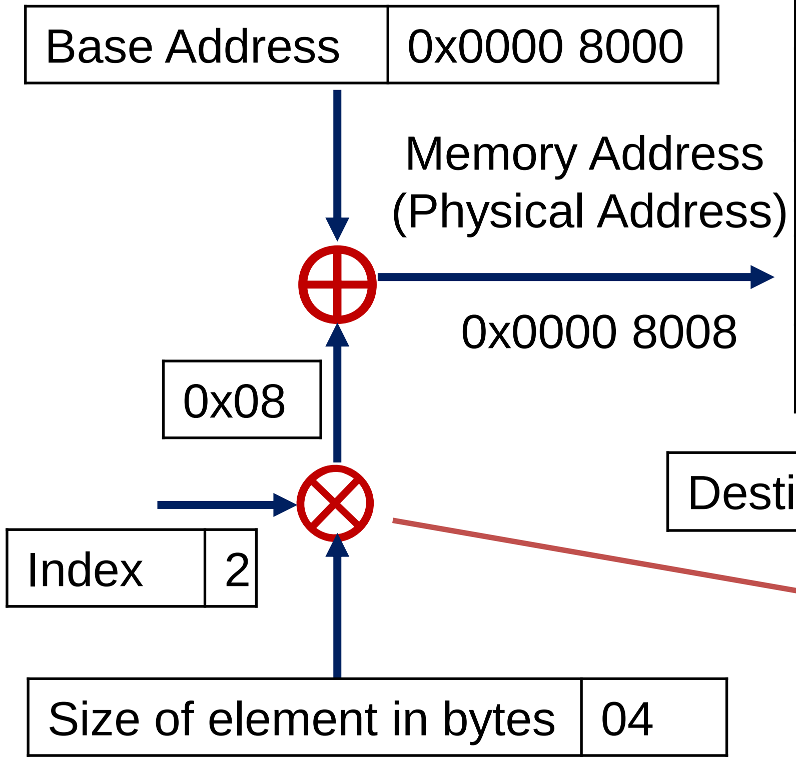
- In RISC V supports The **load-store computer architecture** : Separates instructions into two types of operations.
  - Memory access operations (load and stores) and
  - operations that operate on the data in the register file (register to register or register to immediate).
- **By not combining memory accesses with data manipulation operations, the processor's complexity is reduced which enables making the common case fast,** one of the eight great ideas in computer architecture.
- Therefore, RISC-V must include instructions that transfer data between memory and registers. Such instructions are called **data transfer instructions.**
- **Example : lw, sw, ld, sd and so on**



Operands of the Computer Hardware

2. Memory Operands

How a computer represent and access such complex data **structures?**



Bits → Address	31 24	23 16	15 8	7 0	Element
	Byte3	Byte2	Byte1	Byte0	
0x0000 8000	8F	34	3D	DD	0
0x0000 8004	67	DF	ED	11	1
0x0000 8008	77	66	FF	3D	2
0x0000 800C	11	22	DD	EE	3

Destination Register	77	66	FF	3D

This multiplication can be done by **shifting Index (if known) by 2 times (if data of 32 bit size) towards left.**

a) (0x02) <<2 = (0000 0010)b << 2 = (0000 1000)b= 0x08

Or Manually calculated and then taken as imm12 offset provident element number in the array is known

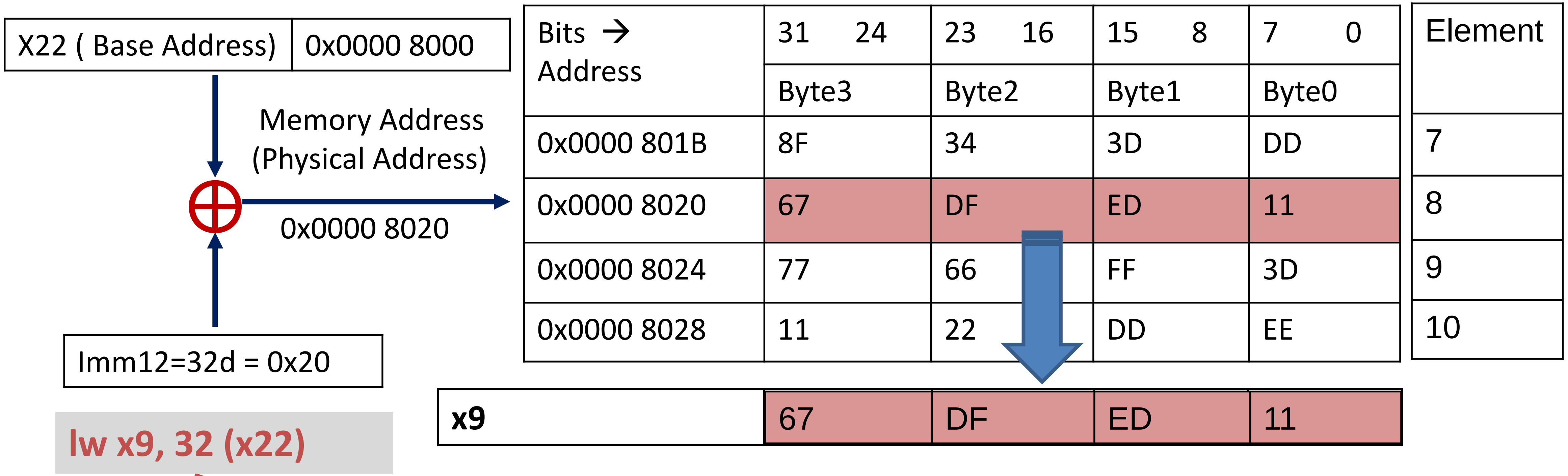
Instructions – Language of Computer

Operands of the Computer Hardware

Saved Registers	x8, x9, x18-x27
Temporary registers	x5-x7, x21-x31

2. Memory Operands

Load Instruction (lw) : Syntax - lw rd,imm12(rs1)



Bits → Address	31 24	23 16	15 8	7 0	Element
	Byte3	Byte2	Byte1	Byte0	
0x0000 801B	8F	34	3D	DD	7
0x0000 8020	67	DF	ED	11	8
0x0000 8024	77	66	FF	3D	9
0x0000 8028	11	22	DD	EE	10

x9	67	DF	ED	11

lw x9, 32 (x22)

The offset can be taken as imm12 by manually calculating
Imm12 = offset = Element Number x 4 = 8 x 4 =32=0x20

Operands of the Computer Hardware

2. Memory Operands

Store Instruction (sw)

Store (sw) : It copies data from a register to memory.

Syntax: sw source reg, offset(base reg)

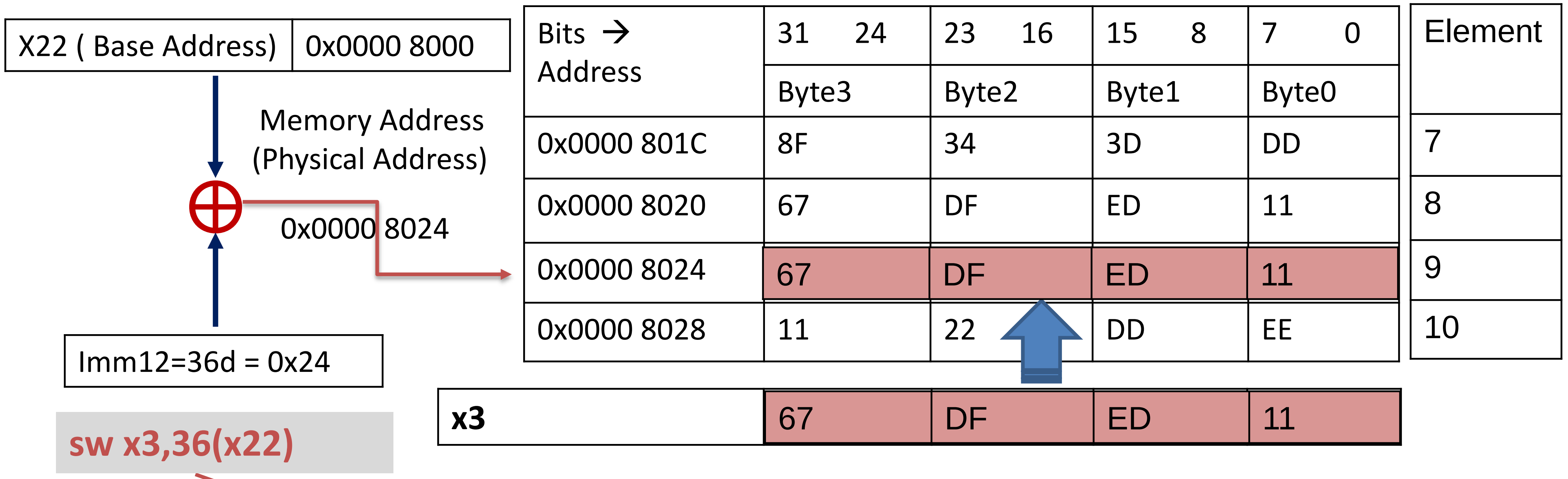
- Once again, the RISC-V address is specified in part by a constant and in part by the contents of a register
- sw stands fore store word

- Example: sw x9,36(x22)

Instructions – Language of Computer
Operands of the Computer Hardware

2. Memory Operands

Store Instruction (sw) : Syntax: sw rs2,imm12(rs1)



Bits → Address	31 24 / Byte3	23 16 / Byte2	15 8 / Byte1	7 0 / Byte0	Element
0x0000 801C	8F	34	3D	DD	7
0x0000 8020	67	DF	ED	11	8
0x0000 8024	67	DF	ED	11	9
0x0000 8028	11	22	DD	EE	10

The offset can be taken as imm12 by manually calculating
Imm12 = offset = Element Number x 4 = 8 x 4 =32=0x20

Instructions – Language of Computer

Operands of the Computer Hardware

2. Memory Operands

What is the RISC-V assembly code for the C assignment statement below?

A[12] = h + A[8];

Assume variable

h	Associated with x21
Base Address of Array A	Associated with x22

x21	h
x22	Base Address of array A

Address Hexadei	Address Decimal	Data (32bits)
0X2020	2032	A[8]
0X201B	2028	A[7]
0X2018	2024	A[6]
0X2014	2020	A[5]
0X2010	2016	A[4]
0X200B	2012	A[3]
0X2008	2008	A[2]
0X2004	2004	A[1]
0X2000	2000	A[0]

Instructions – Language of Computer

Operands of the Computer Hardware

2. Memory Operands

What is the RISC-V assembly code for the C assignment statement below?

g = h + A[8];

Assume variable

g	Associated with x20
h	Associated with x21
Base Address of Array A	Associated with x22

Address Hexadei	Address Decimal	Data (32bits)
0X2020	2032	A[8]
0X201B	2028	A[7]
0X2018	2024	A[6]
0X2014	2020	A[5]
0X2010	2016	A[4]
0X200B	2012	A[3]
0X2008	2008	A[2]
0X2004	2004	A[1]
0X2000	2000	A[0]

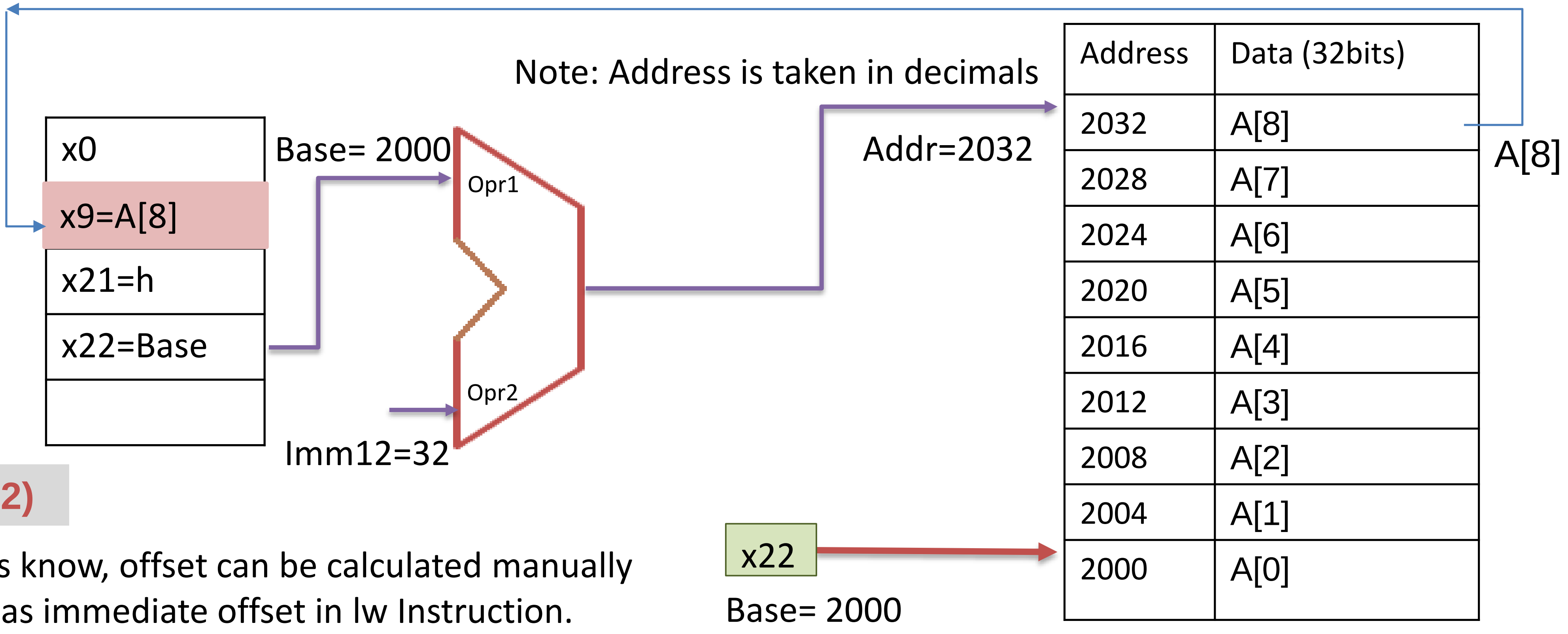
Instructions – Language of Computer

Operands of the Computer Hardware

2. Memory Operands

Assume variable h is associated with register x21 and the base address of the array A is in x22. What is the RISC-V assembly code for the C assignment statement below?

A[12] = h + A[8];



Address	Data (32bits)
2032	A[8]
2028	A[7]
2024	A[6]
2020	A[5]
2016	A[4]
2012	A[3]
2008	A[2]
2004	A[1]
2000	A[0]

```
lw        x9, 32(x22)
```

As element Number is know, offset can be calculated manually (8 x 4 =32) and taken as immediate offset in lw Instruction.

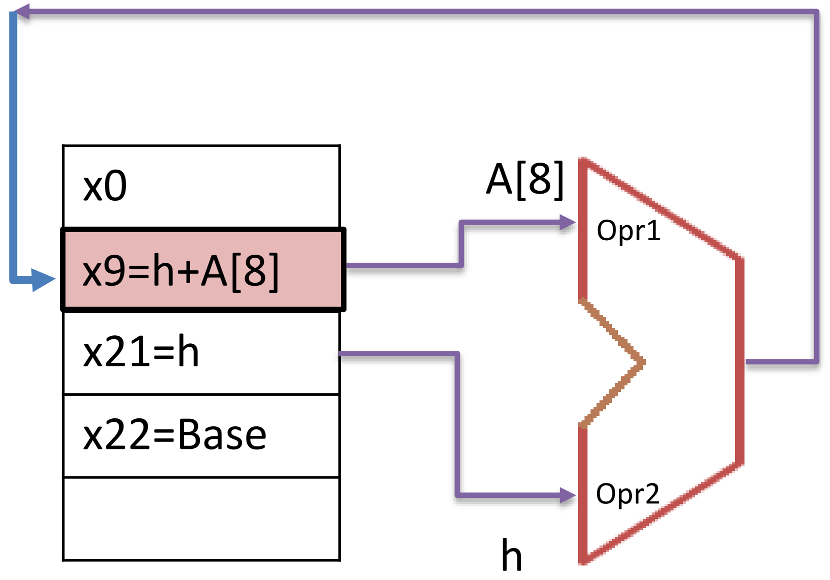
Instructions – Language of Computer

Operands of the Computer Hardware

2. Memory Operands

Assume variable h is associated with register x21 and the base address of the array A is in x22. What is the RISC-V assembly code for the C assignment statement below?

A[12] = h + A[8];



Address	Data (32bits)
2032	A[12]
2028	A[11]
2024	A[10]
2020	A[9]
2032	A[8]
2028	A[7]
2024	A[6]
2020	A[5]
2016	A[4]
2012	A[3]
2008	A[2]
2004	A[1]
2000	A[0]

Base= 2000

x22

```
add        x9, x21,x9
```

Instructions – Language of Computer

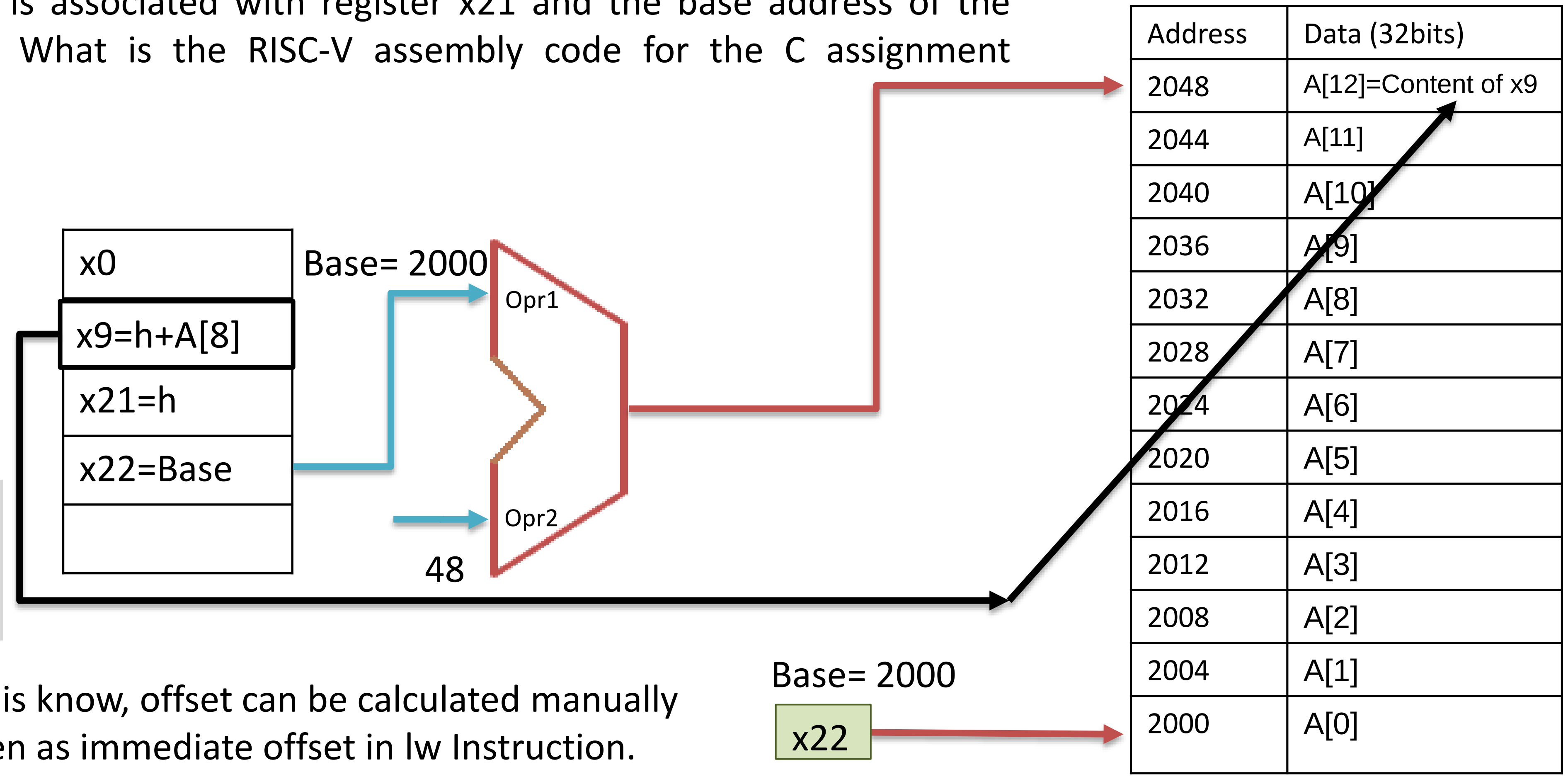
Operands of the Computer Hardware

2. Memory Operands

Assume variable h is associated with register x21 and the base address of the array A is in x22. What is the RISC-V assembly code for the C assignment statement below?

A[12] = h + A[8];

```
lw      x9, 32(x22)
add     x9, x21,x9
sw      x9, 48(x22)
```



Address	Data (32bits)
2048	A[12]=Content of x9
2044	A[11]
2040	A[10]
2036	A[9]
2032	A[8]
2028	A[7]
2024	A[6]
2020	A[5]
2016	A[4]
2012	A[3]
2008	A[2]
2004	A[1]
2000	A[0]

As element Number is know, offset can be calculated manually (12 x 4 =48) and taken as immediate offset in lw Instruction.

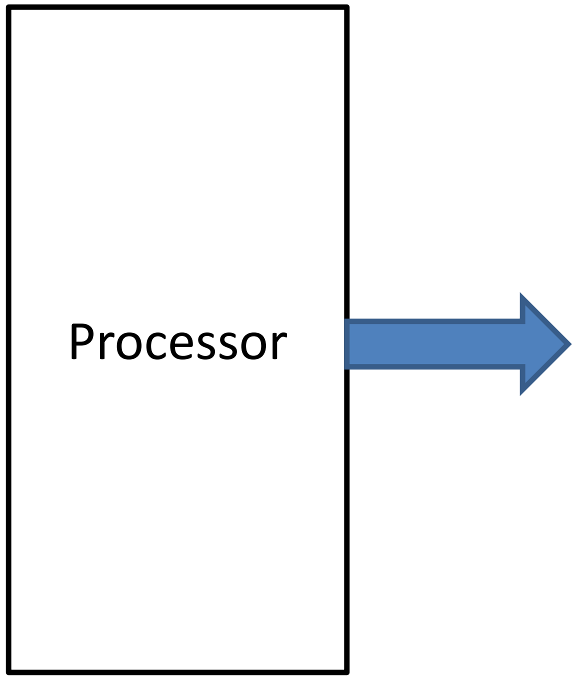
Instructions – Language of Computer

Operands of the Computer Hardware

2. Memory Operands

How RISC V access double word from the Memory?

- RISC-V actually uses byte addressing, with **each double-word representing 8 bytes.**
- **Therefore, the addressing will have an offset of 8 for accessing a double word**



Address	Data
	1010 1010…………………… 10101111
……	
…….	
……	1111 1111…………………… 10101011
24	1000 0001…………………… 11111111
16	0001 0111…………………… 11111111
8	1011 0010…………………… 10101010
0	1011 0101…………………….0000 1111 (64 bits =8 bytes)

Instructions – Language of Computer

Operands of the Computer Hardware

2. Memory Operands

How RISC V access array of 'n' double-word from the Memory?

Compiling an Assignment When an Operand Is in Memory

Example

x20	g
x21	h
x22	Base Address of array A

Let's assume that A is an array of 100 double-words and that the compiler has associated the variables g and **h** with the registers x20 and **x21**. Let's also assume that the starting address, or **base address, of the array is in x22**.

Compile this C assignment statement:
g = h + A[4];

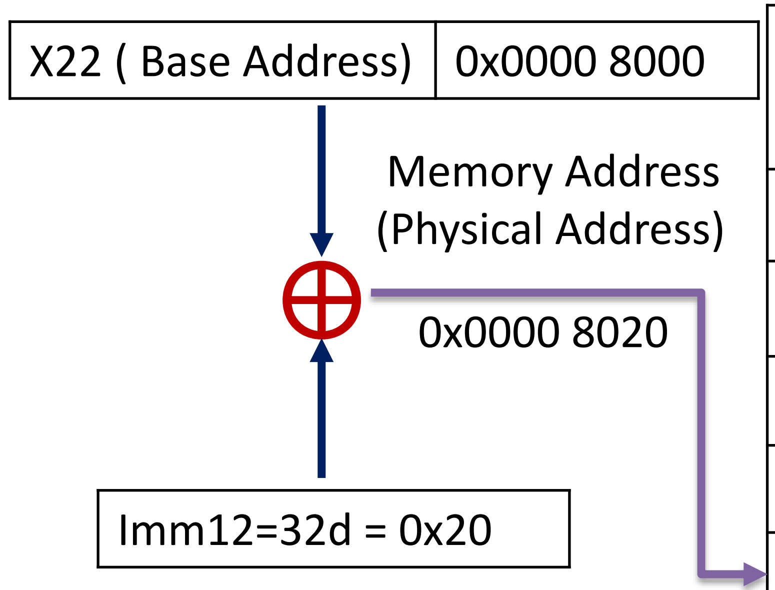
Address (decimal)	Address (Hexadecimal)	Data (64 bits)
2048	2030	A[6]
2040	2028	A[5]
2032	2020	A[4]
2024	2018	A[3]
2016	2010	A[2]
2008	2008	A[1]
2000	2000	A[0]

Instructions – Language of Computer

Operands of the Computer Hardware

2. Memory Operands

Load Instruction (ld) : Syntax - ld rd,imm12(rs1)



ld x9, 32 (x22)

Offset = 4 x 8 = 32

Bits → Address	63 56	55 48	47 40	39 32	31 24	23 16	15 8	7 0	E
	Byte7	Byte6	Byte5	Byte4	Byte3	Byte2	Byte1	Byte0	
0x2000	8F	34	3D	DD	12	78	95	9F	0
0x2008	67	DF	ED	11	8F	34	3D	DD	1
0x2010	77	66	FF	3D	11	22	44	DE	2
0x2018	11	22	DD	EE	05	87	DD	AA	3
0x2020	11	22	DD	EE	FF	3D	11	22	4
0x2028	DD	EE	05	87	8F	34	11	77	5

x9	11	22	DD	EE	FF	3D	11	22

Instructions – Language of Computer

Operands of the Computer Hardware

2. Memory Operands

How RISC V access array of ‘n’ double-word from the Memory?

Compiling an Assignment When an Operand Is in Memory

Example

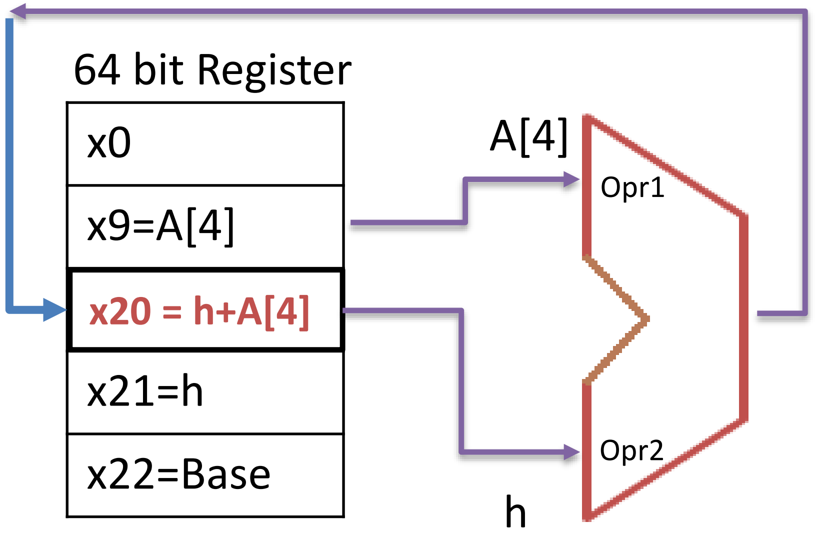
Compile this C assignment statement: **g = h + A[4];**

ld x9, 32(x22)

add x20, x21, x9

x20	g
x21	h
x22	Base Address of array A

Saved Registers	x8, x9, x18-x27
Temporary registers	x5-x7, x21-x31



Instructions – Language of Computer

Operands of the Computer Hardware

2. Constants – Immediate Operand

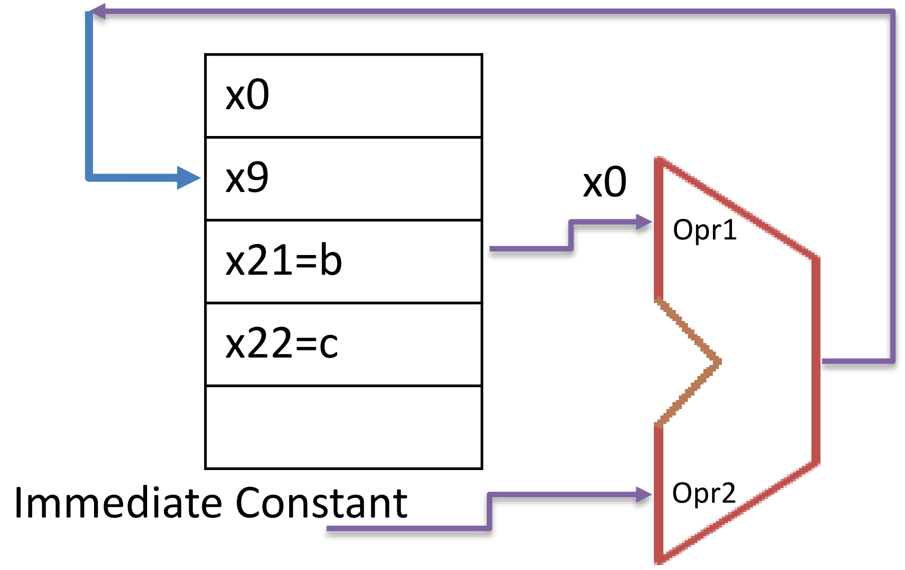
- **Constant data specified in an instruction**
- **Make the common case fast**
  - ✓ **Small constants are common**
  - ✓ Immediate **operand avoids use of Register for a variable / Memory to hold the constants**

C Code

```
// int is a 32-bit signed word
int a = -372;
int b = a + 6;
```

RISC-V assembly code

```
# x0 = a, x9 = b
addi x8, zero, -372
addi x9, x8, 6
```



Note: Any immediate that needs **more than 12 bits** can not use this method.

Instructions – Language of Computer

Operands of the Computer Hardware

Example : Add two numbers present in the memory and store the result Next

```
.data
Array1: .word 0x80000000,0x8000000f,00,0xFFFFFFFF
.text
la x18, Array1 # loads the location address of the specified SYMBOL
lw x8,0(x18)
lw x9,4(x18)
add x19,x8,x9
sw x6,8(x18)
```

Address Hexadeicimal	Data (32bits)
0X1000 000B	Array1[3]
0X1000 0008	Array1[2]
0X1000 0004	Array1[1]
0X1000 0000	Array1[0]

Memory Mapping

0x0000 0000 **0x0FFF FFFF**	**Code Memory**
0x1000 0000 0x1FFF FFFF	Data Memory

THANK YOU

Mahesh Awati
Department of Electronics and Communication

mahesha@pes.edu

+91 9741172822