CE/CZ 3001:
Advanced Computer Architecture

(Module 2: Instruction Set Architecture Design)

Dr Smitha K. G.
School of Computer Science
And Engineering

Outline

- Introduction to ISA

- ARMv8 ISA: A design example named as LEGv8
  - Functionality category
    - Arithmetic and Logical instructions
    - Data transfer instructions
    - Conditional instructions
    - Unconditional instructions

ARMv8 ISA(specifically LEGv8): A design example

Memory and register specification, Instruction format, addressing modes and instruction set.

ARM ISA

ARM (Advanced RISC Machine, originally Acorn RISC Machine)

- Simple, sensible, regular, widely used RISC architecture
- Two major revisions of ARM at present: ARMv7 for 32-bit address and ARMv8 for 64-bit address.
- We shall focus on ARMv8.

LEG(Lessen Extrinsic Garrulity v8): A design example

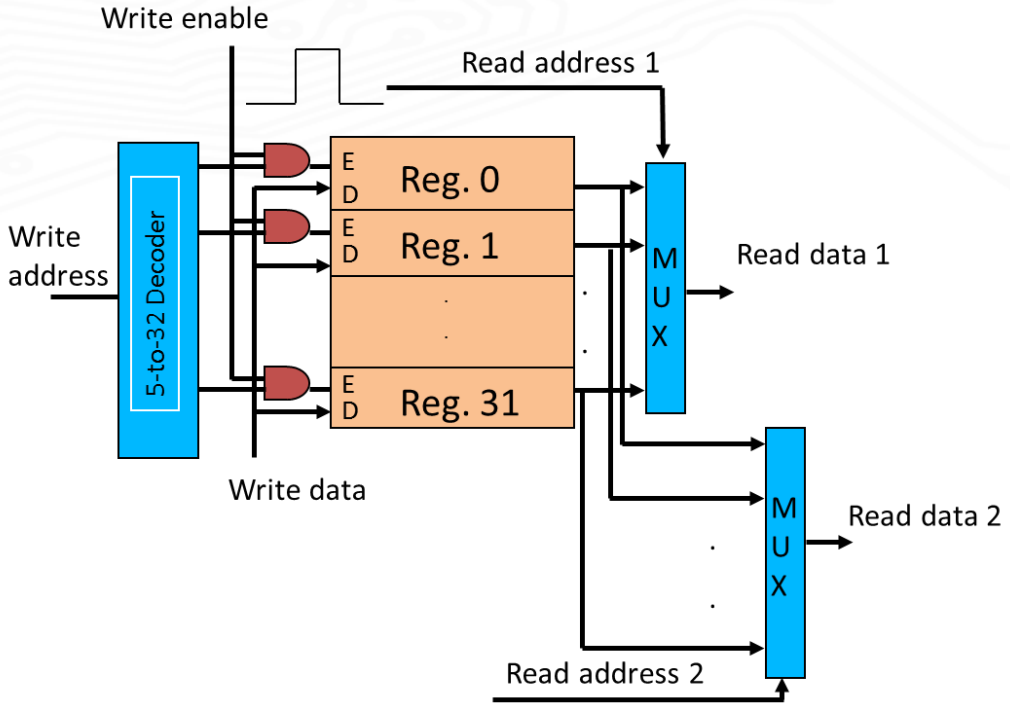
- 64 for bit address bus
- 64-bit data
- 32-bit instruction
- The size of a register in the LEGv8 architecture is 64 bits; groups of 64 bits (8 bytes- doubleword).
- 32 register files with each register storing a 64-bit data

Register specification (Part 1/2)

CPU registers: Used for frequently accessed data

General purpose register (GPR) file

- Contains thirty-two 64-bit registers
- Contains 2 read ports and 1 write port



Q: Why only thirty-two registers only? Can it be more?

A1: More registers will make it significantly slower

A2: More registers will increase the operand size

Program counter (PC):

64-bit register that holds the address of the next instruction

Why PC is incremented by 4 and not 1?

Address	instruction
0x0000000000000000	0xAC
0x0000000000000001	0x1C
0x0000000000000002	0xAB
0x0000000000000003	0x5E
.....	.....
.....	.....
0xFFFFFFFFFFFFFFFC	0x62
0xFFFFFFFFFFFFFFFD	0x03
0xFFFFFFFFFFFFFFFE	0xD3
0xFFFFFFFFFFFFFFFF	0x5A

1 byte

Address	Instruction
0x0000000000000000	0x5EAB1CAC
0x0000000000000004	.....
0x0000000000000008	.....
0x000000000000000C	.....
.....	.....
.....	.....
0xFFFFFFFFFFFFFFF0	.....
0xFFFFFFFFFFFFFFF4	.....
0xFFFFFFFFFFFFFFF8	.....
0xFFFFFFFFFFFFFFFC	0x5AD30362

Little Endian method

- Instruction size (LEGv8) is 32 bits= 4 bytes
- Address of next instruction is 4 more than that of current instruction

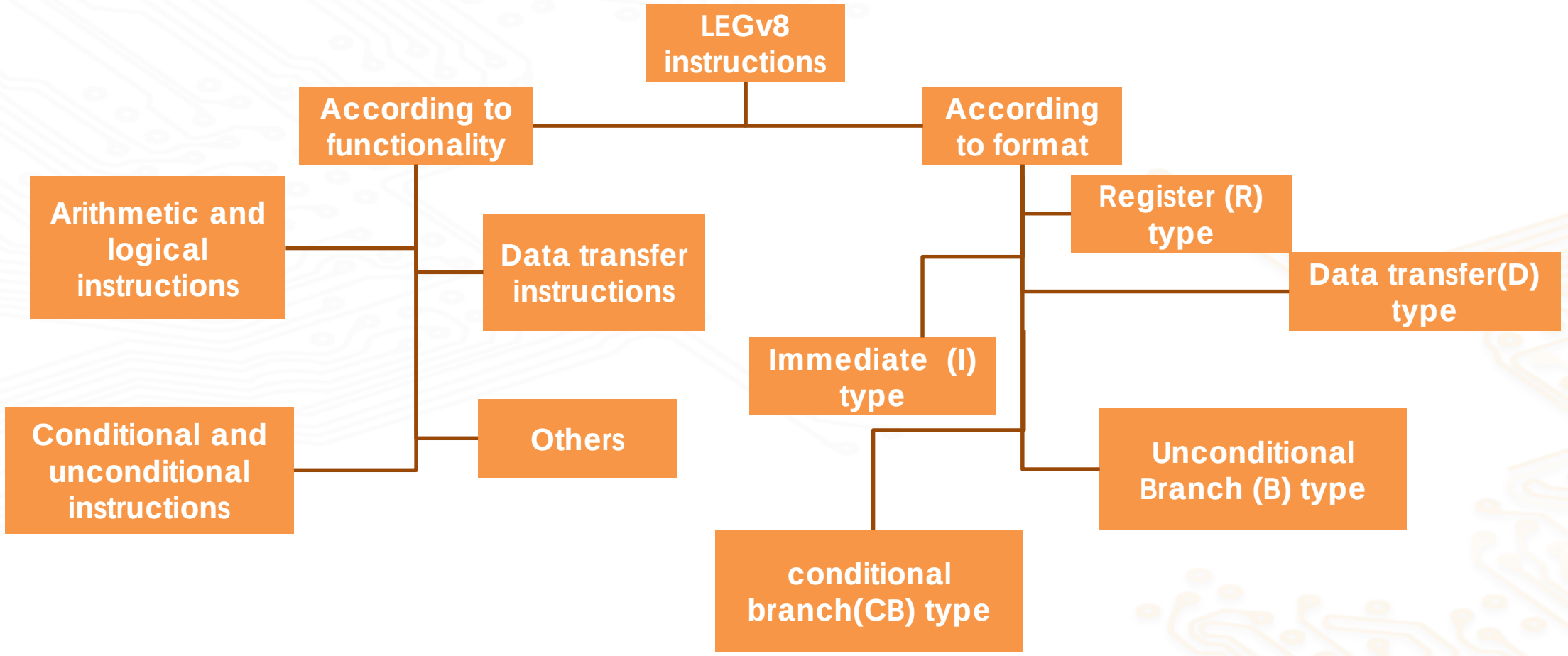
Data Memory organization

Address	Data	Address	data
0x0000000000000000	0xAC	0x....00000	0x5AD303625EAB1CAC
0x0000000000000001	0x1C	0x....00008	.....
0x0000000000000002	0xAB	0x....00010	.....
0x0000000000000003	0x5E	0x....00018	.....
0x0000000000000004	0x62	.....	.....
0x0000000000000005	0x03	.....	.....
0x0000000000000006	0xD3	.....	.....
0x0000000000000007	0x5A	.....	.....
.....	.....	.....	.....
.....	.....	.....	.....

- Data size (in reg file and Dmem)(LEGv8) is 64 bits= 8 bytes
- Address of next 64 bit data from the data memory is 8 more than that of current 64 bit data

Little Endian method

Classification of LEGv8 instructions



ALU instructions – arithmetic operations

arithmetic operations: ADD, SUB, ADDI, SUBI and their variants

Category	InstructionExample		Meaning	Comments
Arithmetic	add	ADD X1, X2, X3	X1 = X2 + X3	Three register operands
	subtract	SUB X1, X2, X3	X1 = X2 – X3	Three register operands
	add immediate	ADDI X1, X2, 20	X1 = X2 + 20	Used to add constants
	subtract immediate	SUBI X1, X2, 20	X1 = X2 – 20	Used to subtract constants
	add and set flags	ADDS X1, X2, X3	X1 = X2 + X3	Add, set condition codes
	subtract and set flags	SUBS X1, X2, X3	X1 = X2 – X3	Subtract, set condition codes
	add immediate and set flags	ADDIS X1, X2, 20	X1 = X2 + 20	Add constant, set condition codes
	subtract immediate and set flags	SUBIS X1, X2, 20	X1 = X2 – 20	Subtract constant, set condition codes

Condition codes, set from arithmetic instruction with S-suffix (ADDS, ADDIS, ANDS, ANDIS, SUBS, SUBIS)

negative (N): result had 1 in MSB

zero (Z): result was 0

overlow (V): result overflowed

carry (C): result had carryout from MSB

ALU – logical operations

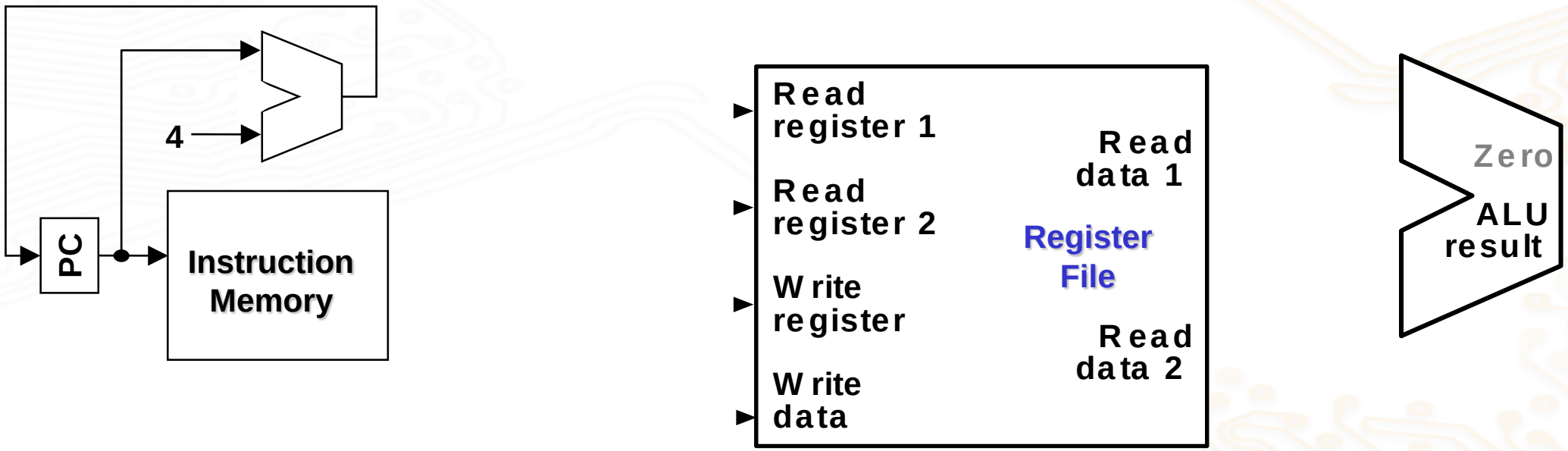
Logical operations: AND, ORR, EOR, ANDI and their variants

Logical	and	AND X1, X2, X3	X1 = X2 & X3	Three reg. operands; bit-by-bit AND
	inclusive or	ORR X1, X2, X3	X1 = X2 | X3	Three reg. operands; bit-by-bit OR
	exclusive or	EOR X1, X2, X3	X1 = X2 ^ X3	Three reg. operands; bit-by-bit XOR
	and immediate	ANDI X1, X2, 20	X1 = X2 & 20	Bit-by-bit AND reg. with constant
	inclusive or immediate	ORRI X1, X2, 20	X1 = X2 | 20	Bit-by-bit OR reg. with constant
	exclusive or immediate	EORI X1, X2, 20	X1 = X2 ^ 20	Bit-by-bit XOR reg. with constant
	logical shift left	LSL X1, X2, 10	X1 = X2 << 10	Shift left by constant
	logical shift right	LSR X1, X2, 10	X1 = X2 >> 10	Shift right by constant

Logical shift left : Shift left and fill with 0 bits
LSL by i bits multiplies by 2^i
Logical shift right: Shift right and fill with 0 bits
LSR by i bits divides by 2^i (unsigned only)

Visual example : Add X2, X1, X0

`fetch` `decode` `writeback` `execute`



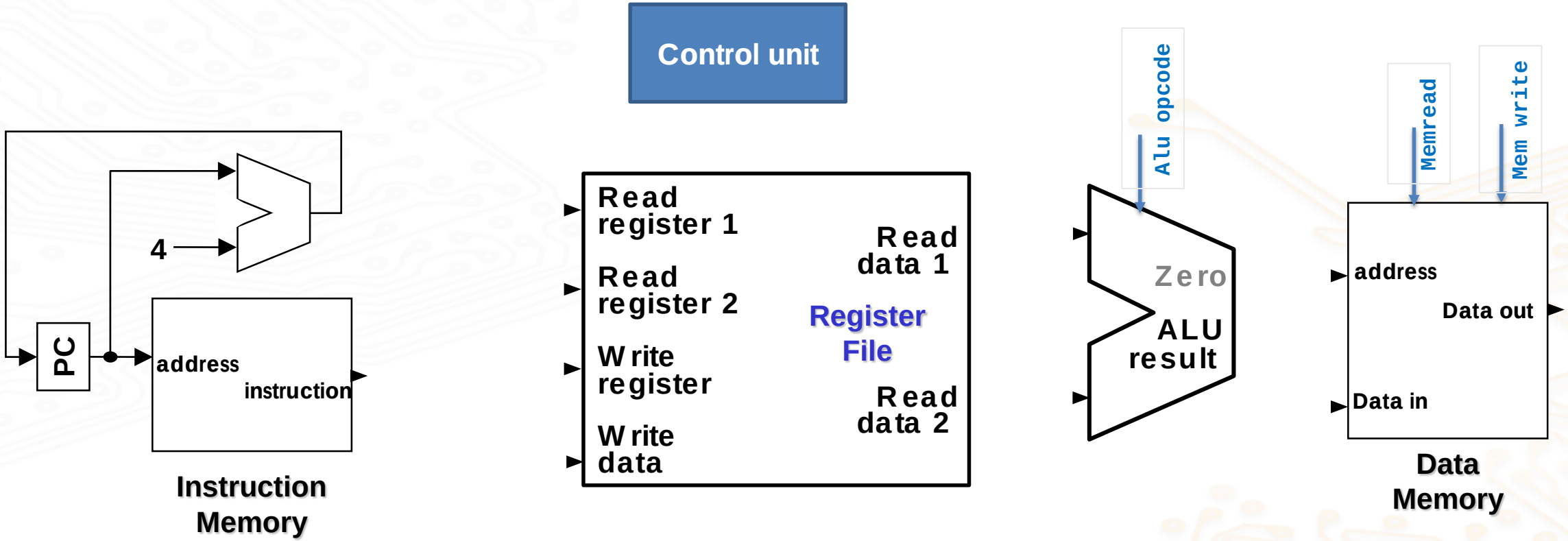
- 64 for bit address bus
- 64-bit data
- 32-bit instruction
- 32 register files each having 64 bit data

Data transfer operations

Data transfer operations: LDUR, STUR, LDURB, STURB

Data transfer	load register	LDUR X1, [X2,40]	X1 = Memory[X2 + 40]	Doubleword from memory to register
	store register	STUR X1, [X2,40]	Memory[X2 + 40] = X1	Doubleword from register to memory
	load signed word	LDURSW X1,[X2,40]	X1 = Memory[X2 + 40]	Word from memory to register
	store word	STURW X1, [X2,40]	Memory[X2 + 40] = X1	Word from register to memory
	load half	LDURH X1, [X2,40]	X1 = Memory[X2 + 40]	Halfword memory to register
	store half	STURH X1, [X2,40]	Memory[X2 + 40] = X1	Halfword register to memory
	load byte	LDURB X1, [X2,40]	X1 = Memory[X2 + 40]	Byte from memory to register
	store byte	STURB X1, [X2,40]	Memory[X2 + 40] = X1	Byte from register to memory

Visual example : LDUR X2, [X1 ,#8]

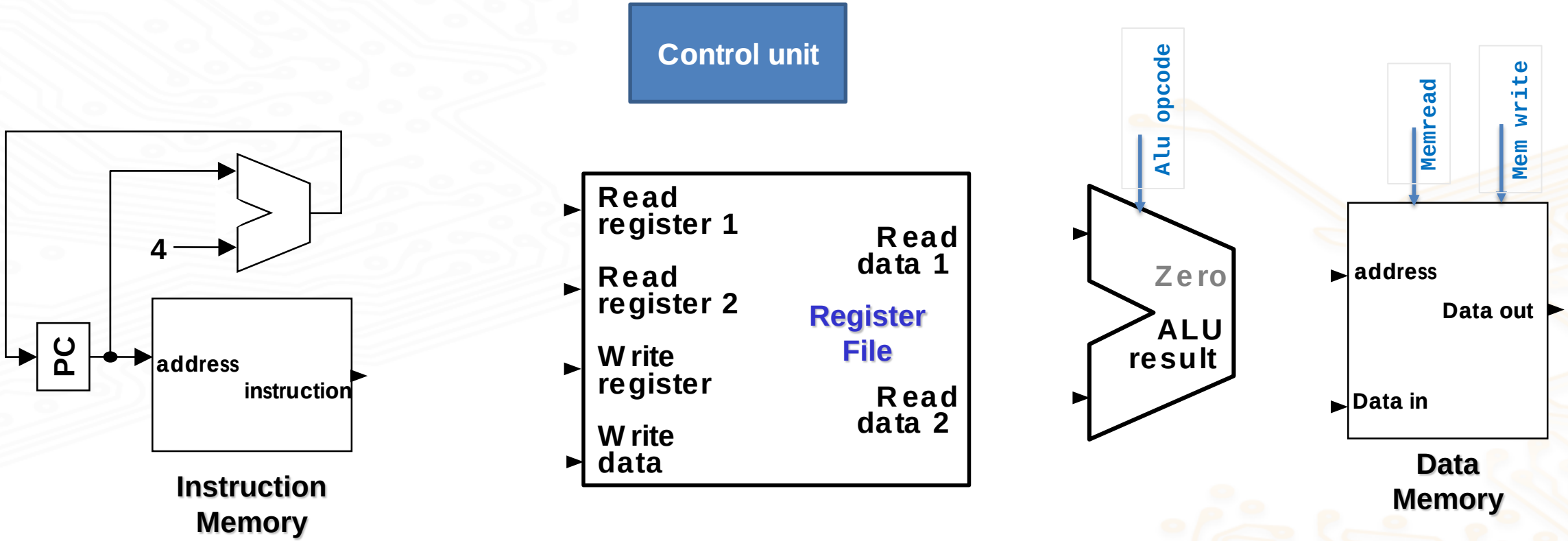


What will be the result to be loaded to register file after execution of LDUR X4, [X2,#4]? Snapshots of DMEM and Registerfile are given below.

Register file
Reg0=0x4
Reg1=0x8
Reg2=0xC
Reg3=0x10
Reg4=0x14
……
…….
Reg31=0x0

Address	data
0x….00000	0x0
0x….00008	0x5
0x….00010	0xA
0x….00018	0xF
……	……
……	……
……	……

Visual example : STUR X2, [X1 ,#8]



What will be the result after execution of
STUR X4, [X2,#4]? Snapshots of DMEM and Registerfile are given below.

Register file
Reg0=0x4
Reg1=0x8
Reg2=0xC
Reg3=0x10
Reg4=0x14
……
…….
Reg31=0x0

Address	data
0x….00000	0x0
0x….00008	0x5
0x….00010	0xA
0x….00018	0xF
……	……
……	……
……	……

Conditional and unconditional operations

Control Flow operations: **B, B. cond, CBZ, CBNZ, BR, BL**

Conditional branch	compare and branch on equal 0	`CBZ X1, 25`	`if (X1 == 0) go to PC + 100`	Equal 0 test; PC-relative branch
	compare and branch on not equal 0	`CBNZ X1, 25`	`if (X1 != 0) go to PC + 100`	Not equal 0 test; PC-relative branch
	branch conditionally	`B.cond 25`	`if (condition true) go to PC + 100`	Test condition codes; if true, branch
Unconditional branch	branch	`B 2500`	`go to PC + 10000`	Branch to target address; PC-relative
	branch to register	`BR X30`	`go to X30`	For switch, procedure return
	branch with link	`BL 2500`	`X30 = PC + 4; PC + 10000`	For procedure call PC-relative

Summary

- Introduction to ISA

- ARMv8 ISA: A design example named as LEGv8
  - Functionality category
    - Arithmetic and Logical instructions
    - Data transfer instructions
    - Conditional instructions
    - Unconditional instructions