L4_1 ARM-ISA_Arithmetic-Logical_Instructions

Assignment Project Exam Help
https://powcoder.com

EECS 370 – Introduction to Computer Organization – Fall 2020

Add WeChat powcoder

Learning Objectives

- Recognize the set of instructions for ARM Architecture (ISA) and be able to describe the operations and operands for instructions
  - LEGv8 subset
- Ability to create simple ARM assembly programs, e.g., using mathematical, logic, and memory operations

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

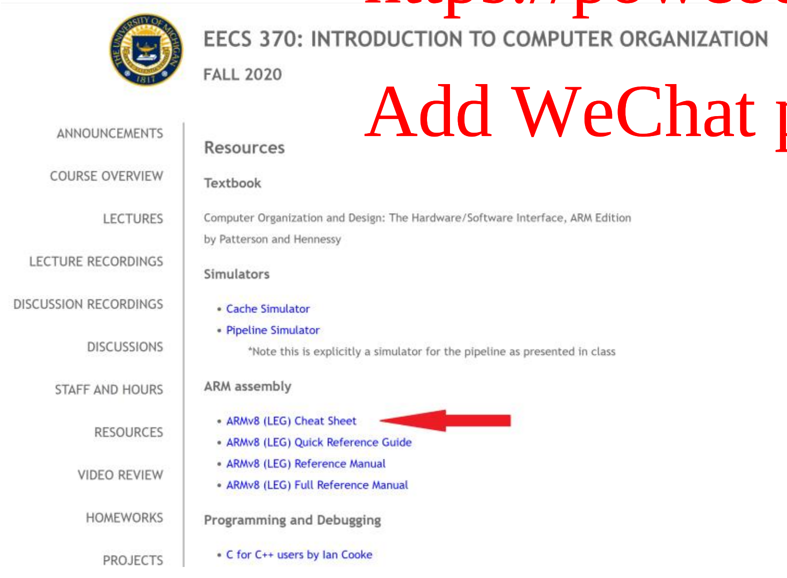
Resources

- Many resources on 370 website
  - https://www.eecs.umich.edu/courses/eecs370/eecs370.f20/resources/
    - ARMv8 references
- Discussion recordings
- Piazza
- Office hours

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder



EECS 370: INTRODUCTION TO COMPUTER ORGANIZATION

FALL 2020

ANNOUNCEMENTS
COURSE OVERVIEW
LECTURES
LECTURE RECORDINGS
DISCUSSION RECORDINGS
DISCUSSIONS
STAFF AND HOURS
RESOURCES
VIDEO REVIEW
HOMEWORKS
PROJECTS

Resources

Textbook

Computer Organization and Design: The Hardware/Software Interface, ARM Edition by Patterson and Hennessy

Simulators

- Cache Simulator
- Pipeline Simulator

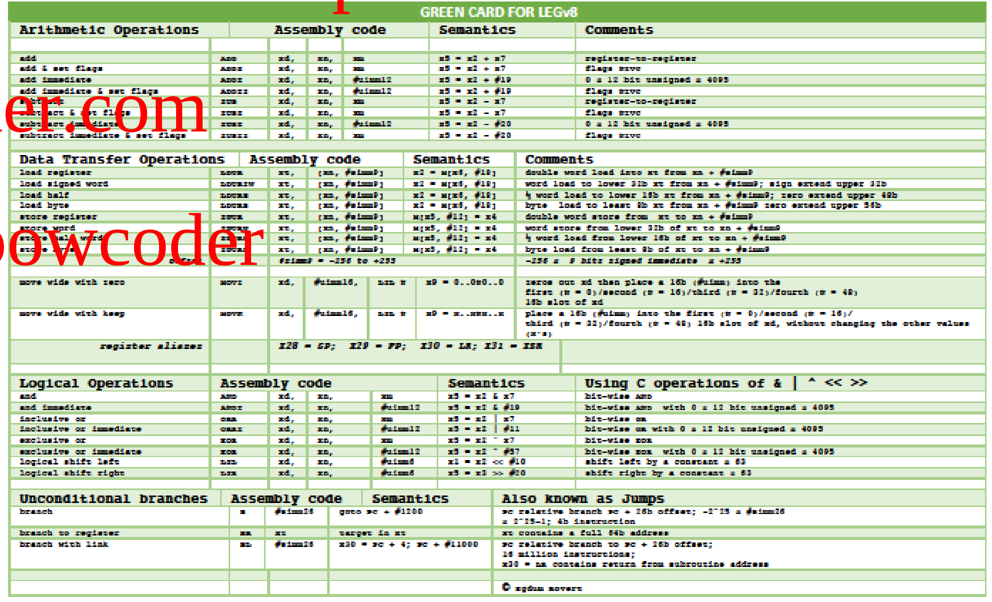
*Note this is explicitly a simulator for the pipeline as presented in class

ARM assembly

- ARMv8 (LEG) Cheat Sheet
- ARMv8 (LEG) Quick Reference Guide
- ARMv8 (LEG) Reference Manual
- ARMv8 (LEG) Full Reference Manual

Programming and Debugging

- C for C++ users by Ian Cooke



GREEN CARD FOR LEGv8

Arithmetic Operations	Assembly code				Semantics	Comments
add	ADD	xd,	xn,	xm	x5 = x2 + x7	register-to-register
add & set flags	ADDS	xd,	xn,	xm	x5 = x2 + x7	flags NZVC
add immediate	ADDI	xd,	xn,	#uimm12	x5 = x2 + #19	0 ≤ 12 bit unsigned ≤ 4095
add immediate & set flags	ADDIS	xd,	xn,	#uimm12	x5 = x2 + #19	flags NZVC
subtract	SUB	xd,	xn,	xm	x5 = x2 - x7	register-to-register
subtract & set flags	SUBS	xd,	xn,	xm	x5 = x2 - x7	flags NZVC
subtract immediate	SUBI	xd,	xn,	#uimm12	x5 = x2 - #20	0 ≤ 12 bit unsigned ≤ 4095
subtract immediate & set flags	SUBIS	xd,	xn,	xm	x5 = x2 - #20	flags NZVC

Data Transfer Operations	Assembly code				Semantics	Comments
load register	LDUR	xt,	[xn, #simm9]		x2 = M[x8, #16]	double word load into xt from xn + #simm9
load signed word	LDURSW	xt,	[xn, #simm9]		x2 = M[x8, #16]	word load to lower 32b xt from xn + #simm9; sign extend upper 32b
load half	LDURH	xt,	[xn, #simm9]		x2 = M[x8, #16]	½ word load to lower 16b xt from xn + #simm9; zero extend upper 48b
load byte	LDURB	xt,	[xn, #simm9]		x2 = M[x8, #16]	byte load to least 8b xt from xn + #simm9 zero extend upper 56b
store register	STUR	xt,	[xn, #simm9]		M[x5, #12] = x4	double word store from xt to xn + #simm9
store word	STURW	xt,	[xn, #simm9]		M[x5, #12] = x4	word store from lower 32b of xt to xn + #simm9
store half word	STURH	xt,	[xn, #simm9]		M[x5, #12] = x4	½ word load from lower 16b of xt to xn + #simm9
store byte	STURB	xt,	[xn, #simm9]		M[x5, #12] = x4	byte load from least 8b of xt to xn + #simm9
offset		#simm9 = -256 to +255				-256 ≤ 9 bits signed immediate ≤ +255
move wide with zero	MOVZ	xd,	#uimm16,	LSL N	x9 = 0..0n0..0	zeros out xd then place a 16b (#uimm) into the first (N = 0)/second (N = 16)/third (N = 32)/fourth (N = 48) 16b slot of xd
move wide with keep	MOVK	xd,	#uimm16,	LSL N	x9 = x..xnx..x	place a 16b (#uimm) into the first (N = 0)/second (N = 16)/third (N = 32)/fourth (N = 48) 16b slot of xd, without changing the other values (x's)
register aliases		X28 = SP; X29 = FP; X30 = LR; X31 = XZR				

Logical Operations	Assembly code				Semantics	Using C operations of & | ^ << >>
and	AND	xd,	xn,	xm	x5 = x1 & x7	bit-wise AND
and immediate	ANDI	xd,	xn,	#uimm12	x5 = x1 & #19	bit-wise AND with 0 ≤ 12 bit unsigned ≤ 4095
inclusive or	ORR	xd,	xn,	xm	x5 = x1 | x7	bit-wise OR
inclusive or immediate	ORRI	xd,	xn,	#uimm12	x5 = x1 | #11	bit-wise OR with 0 ≤ 12 bit unsigned ≤ 4095
exclusive or	EOR	xd,	xn,	xm	x5 = x2 ^ x7	bit-wise XOR
exclusive or immediate	EORI	xd,	xn,	#uimm12	x5 = x2 ^ #57	bit-wise XOR with 0 ≤ 12 bit unsigned ≤ 4095
logical shift left	LSL	xd,	xn,	#uimm6	x1 = x2 << #10	shift left by a constant ≤ 63
logical shift right	LSR	xd,	xn,	#uimm6	x5 = x3 >> #20	shift right by a constant ≤ 63

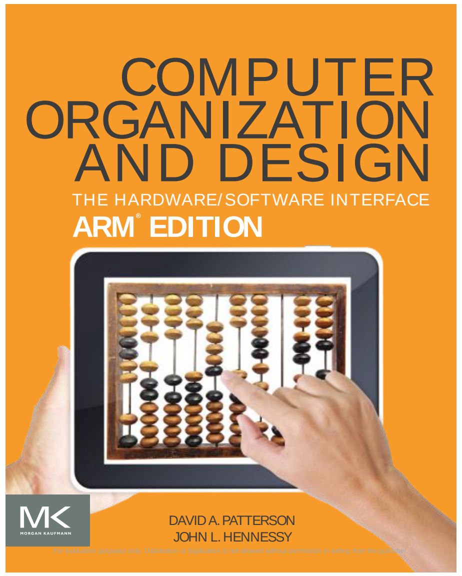
Unconditional branches	Assembly code		Semantics	Also known as Jumps
branch	B	#simm26	goto pc + #1200	pc relative branch pc + 26b offset; -2^25 ≤ #simm26 ≤ 2^25-1; 4b instruction
branch to register	BR	xt	target in xt	xt contains a full 64b address
branch with link	BL	#simm26	x30 = pc + 4; pc + #11000	pc relative branch to pc + 26b offset; 16 million instructions; x30 = LR contains return from subroutine address
				© [illegible]

ARM ISA

ARMs and LEGs

- ARMv8 is the 64 bit version—all registers are 64 bits wide
- Addresses are calculated to be 64 bits too
- BUT: Instructions are 32 bits
- We use a (small) subset of the v8 ISA used in P+H
- It is referred to as the LEGv8 in keeping with the body part theme!

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder


COMPUTER ORGANIZATION AND DESIGN
THE HARDWARE/SOFTWARE INTERFACE
ARM EDITION
DAVID A. PATTERSON
JOHN L. HENNESSY

ARM Instruction Set—LEGv8 subset

ARM ISA

- The main types of instructions fall into the familiar classes we saw with LC-2K:
  1. Arithmetic
     - Add, subtract, multiply (not in LEGv8)
  2. Data transfer
     - Loads a stores—LDUR (load unscaled register), STUR, etc.
  3. Logical
     - AND, ORR, EOR, etc.
     - Logical shifts, LSL, LSR
  4. Conditional branch
     - CBZ, CBNZ, B.cond
  5. Unconditional branch (jumps)
     - B, BR, BL

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

ARM ISA

LEGv8 Arithmetic Instructions

- Format: three operand fields
  - Destination register usually the first one – *check instruction format*
  - `ADD X3, X4, X7` – `Think ADD X3 = X4, X7`
    - LC-2K generally has the destination on the right!!!!
    - e.g. add `1 2 3 // r3 = r1 + r2`
- C-code example: `f = (g + h) - (i + j)`

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Register to variable mapping
X3→g
X4→h
X5→i
X6→j

Map to ARM (simple) operations
`ADD t0, g, h`
`ADD t1, i, j`
`SUB f, t0, t1`

$\longrightarrow$

Map operands to registers
`ADD X1, X3, X4`
`ADD X2, X5, X6`
`SUB X7, X1, X2`

ARM ISA

LEGv8 R-instruction Fields

opcode	Rm	shamt	Rn	Rd
11 bits	5 bits	6 bits	5 bits	5 bits

- Register-to-register operations
- Consider ADD `X3, X4, X7`
  - `R[Rd] = R[Rn] + R[Rm]`
  - `Rd = X3, Rn = X4, Rm = X7`
- Rm = second register operand
- `shamt` = shift amount
  - not used in LEG for ADD/SUB and set to 0
- Rn = first register operand
- Rd = destination register
- ADD opcode is `10001011000`, what are the other fields?

opcode 10001011000

Rm = X7 = 00111

Rn = x4 = 00100

shamt = 0 = 000000

Rd = x3 = 00011

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

ARM ISA

LEGv8 Arithmetic Operations

- Machine State—more on this concept as our understanding evolves
  1. Registers: 32 registers, 64-bit wide. X31 aliased to XZR which is always 0 - cannot use as a destination
  2. PC—Program counter
  3. FLAGS: NZVC – record the results of (arithmetic) operations Negative, Zero, oVerflow, Carry—*not present in LC2K*

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

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

ARM ISA

I-instruction fields

opcode	immediate	Rn	Rd
10 bits	12 bits	5 bits	5 bits

- Format: second source operand can be a register or Immediate—a constant in the instruction itself
  - e.g., `ADDI X3, X4, #10`

Assignment Project Exam Help

- Format: 12 bits for immediate constants `0-4095`

https://powcoder.com

- Do not need negative constants because we have `SUBI`

Add WeChat powcoder

C-code instructions	ARM LEGv8
`f = g + 10`	`ADDI X7, X5, #10`
`f = g - 10`	`SUBI X7, X5, #10`

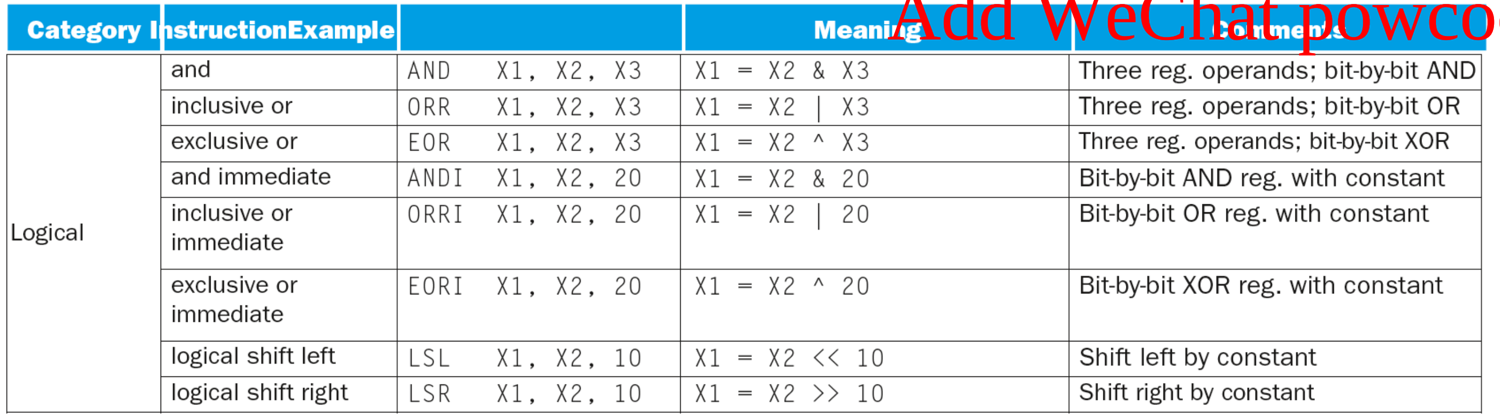
ARM ISA

LEGv8 Logical Instructions

- Logical operations are *bit-wise*
- For example assume
  - `X9 =  11111111 11111111 11111111 00000000 00000000 00000000 00001101 11000000`
  - `X14 = 00000000 00000000 11011010 00000000 00000000 00000000 00111100 00000000`

  `AND X2, X13, X9` would result in
  - `X2 =  00000000 00000000 11011010 00000000 00000000 00000000 00001100 00000000`
- AND and OR correspond to C operators & and |
- For immediate fields the 12-bit constant is padded with zeros to the left—zero extended

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder



Category	Instruction	Example	Meaning	Comments
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

ARM ISA

LEGv8 Shift Logical Instructions

opcode	Rm	shamt	Rn	Rd
11 bits	5 bits	6 bits	5 bits	5 bits

-
  ```
  LSR X6, X23, #2
  X23 = 11111111 11111111 11111111 00000000 00000000 00000000 11011010 00000010
  X6  = 00111111 11111111 11111111 11000000 00000000 00000000 00110110 10000000
  ```
- C-code equivalent : `X6 = X23 >> 2;`

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

- Question: LSL X6, X23, #2 ?
  - What register gets modified?
  - What does it contain after executing the LSL instruction?

- In shift operations Rm is always `0`—`shamt` is 6-bit unsigned

Shifting right by one bit -> divide by 2
Shifting left by one bit -> multiple by 2

ARM ISA

LEGv8 Shift Logical Instructions

opcode	Rm	shamt	Rn	Rd
11 bits	5 bits	6 bits	5 bits	5 bits

- `LSR X6, X23, #2`

```
X23 = 11111111 11111111 11111111 00000000 00000000 00000000 11011010 00000010
X6  = 00111111 11111111 11111111 11000000 00000000 00000000 00110110 10000000
```

Assignment Project Exam Help

- C-code equivalent : `X6 = X23 >> 2;`

https://powcoder.com

- Question: LSL X6, X23, #2 ?
  - What register gets modified?
  - What does it contain after executing the LSL instruction?

Add WeChat powcoder

```
LSL X6, X23, #2
X23 = 11111111 11111111 11111111 00000000 00000000 00000000 11011010 00000010
X6  = 11111111 11111111 11111100 00000000 00000000 00000011 01101000 00001000
```

- In shift operations Rm is always `0`—`shamt` is 6-bit unsigned

Shifting right by one bit -> divide by 2
Shifting left by one bit -> multiple by 2

Pseudo Instructions

- Instructions that use a shorthand “mnemonic” that expands to an assembly instruction
  - Exception to the "1-1 correspondence between assembly and MC" rule

Assignment Project Exam Help

https://powcoder.com

- Example:
  - `MOV  X12, X2 // the contents of X2 copied to X12 – X2 unchanged`

Add WeChat powcoder

- This gets expanded to:
  - `ORR X12, XZR, X2`

- What alternatives could we use instead of ORR?

Pseudo Instructions

- Instructions that use a shorthand "mnemonic" that expands to an assembly instruction
  - Exception to the "1-1 correspondence between assembly and MC" rule

Assignment Project Exam Help

https://powcoder.com

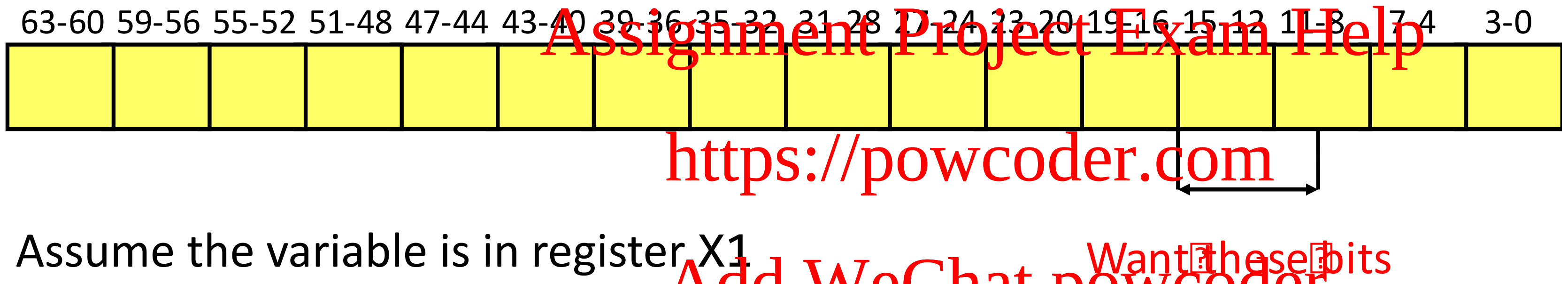
Add WeChat powcoder

- Example:
  - `MOV  X12, X2 // the contents of X2 copied to X12 – X2 unchanged`
- This gets expanded to:
  - `ORR X12, XZR, X2`
- What alternatives could we use instead of ORR? – `ADD X12, ZXR, X2`

LEGv8 Assembly Example #1

ARM ISA

- Show the C and LEGv8 assembly for extracting the value in bits 15:10 from a 64-bit integer variable

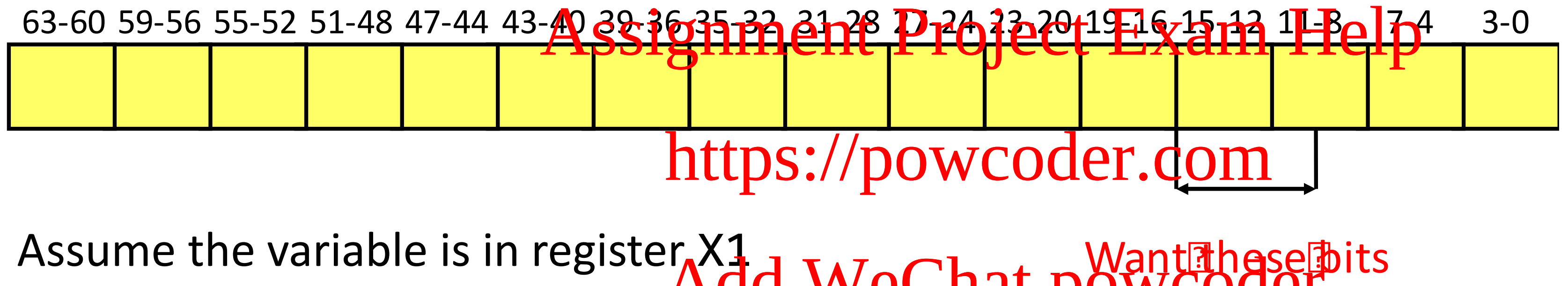


Assume the variable is in register X1

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

LEGv8 Assembly Example #1

- Show the C and LEGv8 assembly for extracting the value in bits 15:10 from a 64-bit integer variable

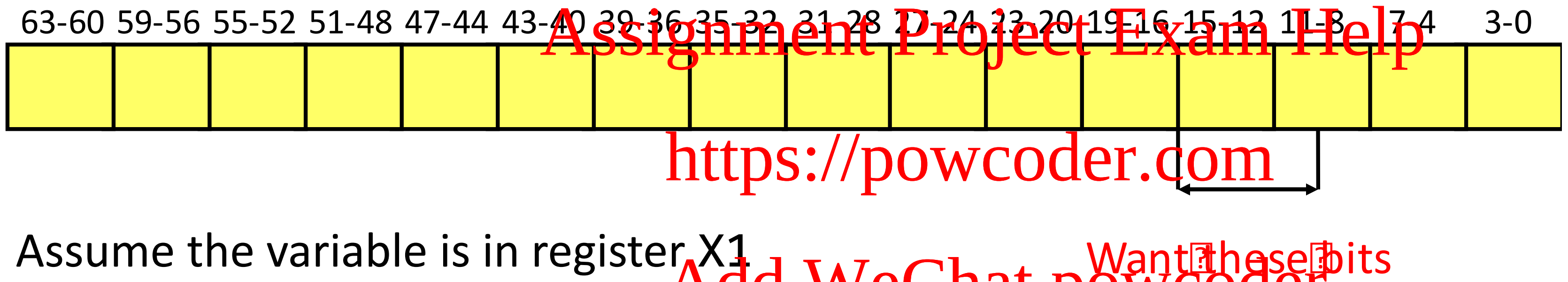


Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Assume the variable is in register X1

LEGv8 Assembly Example #1

- Show the C and LEGv8 assembly for extracting the value in bits 15:10 from a 64-bit integer variable



Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Assume the variable is in register X1

C-code instructions	ARM LEGv8
`x = x >> 10;`	`LSR X1, X1, #10`
`x = x & 0x3F`	`ANDI X1, X1, #0x3F`

Logistics

- There are 4 videos for lecture 4
  - L4_1 – ARM-ISA_Arithmetic-Logical_Instructions
  - L4_2 – ARM-ISA_Memory-Instructions
  - L4_3 – ARM-ISA_Memory-Instructions_Examples
  - L4_4 – C-to-Assembly
- There are two worksheets for lecture 4
  1. LEGv8 Assembly
  2. C to Assembly

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

L4_2 ARM-ISA_Memory-Instructions

Assignment Project Exam Help

https://powcoder.com

EECS 370 – Introduction to Computer Organization – Fall 2020

Add WeChat powcoder

Learning Objectives

- Recognize the set of instructions for ARM Architecture (ISA) and be able to describe the operations and operands for instructions
- Ability to create simple ARM assembly programs, e.g., using mathematical, logic, and memory operations

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

ARM ISA

Word vs Byte Addressing

- A **word** is a collection of bytes
  - Exact size depends on architecture
  - in LC-2K and ARM, 4 bytes
    - **Double word** is 8 bytes
- LC-2K is **word addressable**
  - Each address refers to a particular **word** in memory
  - Want to move forward one int? Increment address by **one**
  - Want move forward one char? Uhhh… no
- ARM (and most modern ISAs) is **byte addressable**
  - Each address refers to a particular **byte** in memory
  - Want to move forward one `int`? Increment address by **four**
  - Want to move forward one `char`? Increment address by **one**

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

ARM ISA

LEGv8 Memory Instructions

- Employs base + displacement addressing mode
  - Base is a register
  - Displacement is 9-bit immediate ±256 bytes
    - Load signed word will sign extended to 64 bits
    - Load half and load byte will zero extend

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

Category	Instruction	Example	Meaning	Comments
	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
	move wide with zero	MOVZ X1,20, LSL 0	X1 = 20 or 20 * 2^{16} or 20 * 2^{32} or 20 * 2^{48}	Loads 16-bit constant, rest zeros
	move wide with keep	MOVK X1,20, LSL 0	X1 = 20 or 20 * 2^{16} or 20 * 2^{32} or 20 * 2^{48}	Loads 16-bit constant, rest unchanged

D-Instruction fields

opcode	address	op2	Rn	Rt
11 bits	9 bits	2 bits	5 bits	5 bits

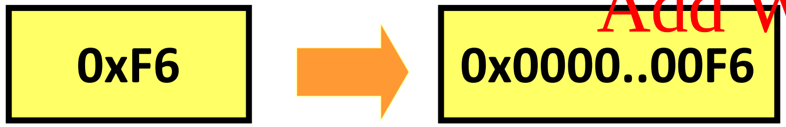
- **D**ata transfer
- opcode and op2 define data transfer operation
  - address is the ±256 bytes displacement
- Rn is the base register
- Rt is the destination (loads) or source (stores)
- More complicated modes are available in full ARMv8

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

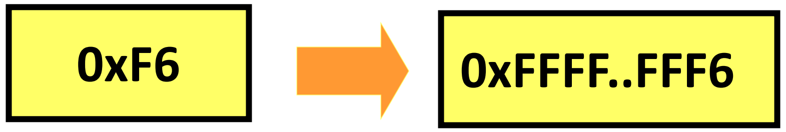
ARM ISA

LEGv8 Memory Instructions

- Registers are 64 bits wide
- But sometimes we want to deal with non-64-bit entities
  - e.g. ints (32 bits), chars (8 bits)
- When we load smaller elements from memory, what do we set the other bits to?
  - Option A: set to zero – LEGv8 instructions LDURH, LDURB



  - Option B: sign extend – LEGv8 instruction LDURSW



Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Load Instruction Sizes

How much data is retrieved from memory at the given address?

- `LDUR X3, [X4, #100]`
  - Load (unscaled) to register—retrieve a double word (64 bits) from address (X4+100)
- `LDURH  X3, [X4, #100]`
  - Load halfword (16 bits) from address (X4+100) to the low 16 bits of X3—top 48 bits of X3 are set zero
- `LDURB X3, [X4, #100]`
  - Load byte (8 bits) from address (X4+100) and put in the low 8 bits of X3—zero extend the destination register X3 (top 56 bits)
- What about loading words?
- `LDURSW X3, [X4, #100]`
  - retrieve a word (32 bits) from address (X4+100) and put in lower half of X3—top 32 bits of X3 are sign extended

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

LEGv8 Data Transfer Instructions--Stores

- Store instructions are simpler—there is no sign/zero extension to consider
- `STUR X3, [X4, #100]`
  - Store (unscaled) register—write the double word (64 bits) in register X3 to the 8 bytes at address (X4+100)
- `STURW X3, [X4, #100]`
  - Store word—write the word (32 bits) in the low 4 bytes of register X3 to the 2 bytes at address (X4+100)
- `STURH X3, [X4, #100]`
  - Store half word—write the half word (16 bits) in the low 2 bytes of register X3 to the 2 bytes at address (X4+100)
- `STURB X3, [X4, #100]`
  - Store byte—write the least significant byte (8 bits) in register X3 to the byte at address (X4+100)

Assignment Project Exam Help

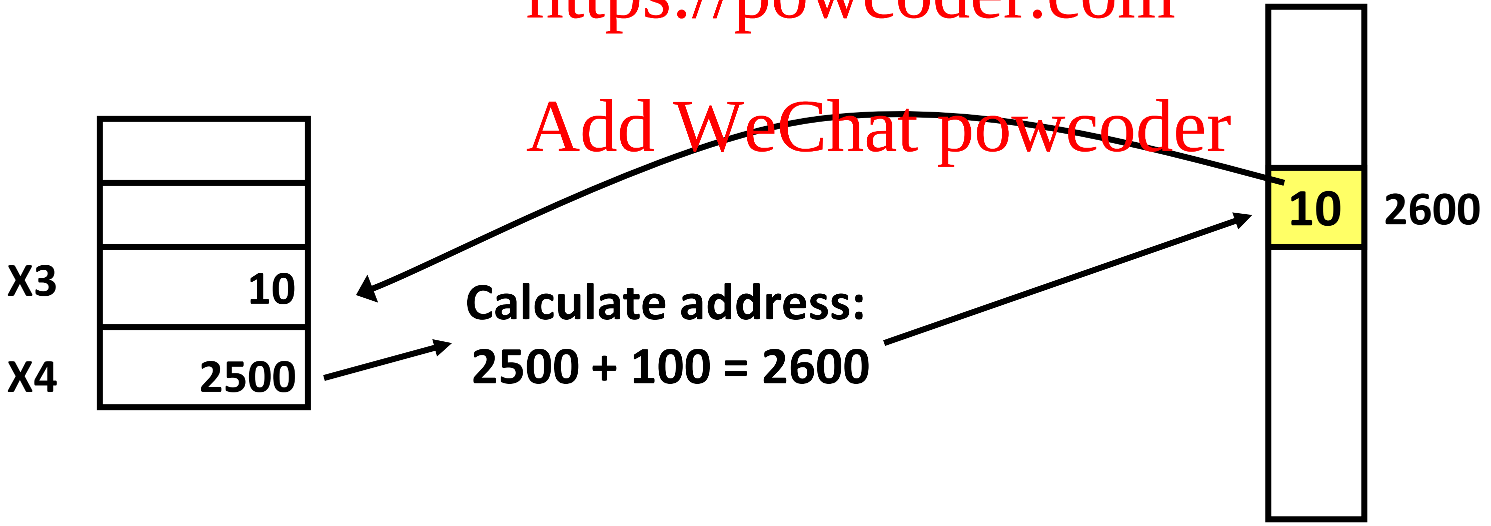
https://powcoder.com

Add WeChat powcoder

Load Instruction in Action

- `LDURB X3, [X4, #100]   // load byte`
  Retrieves 8-bit value from memory location (X4+100) and puts the result into X3. The other 56 most significant bits are 0—zero extended

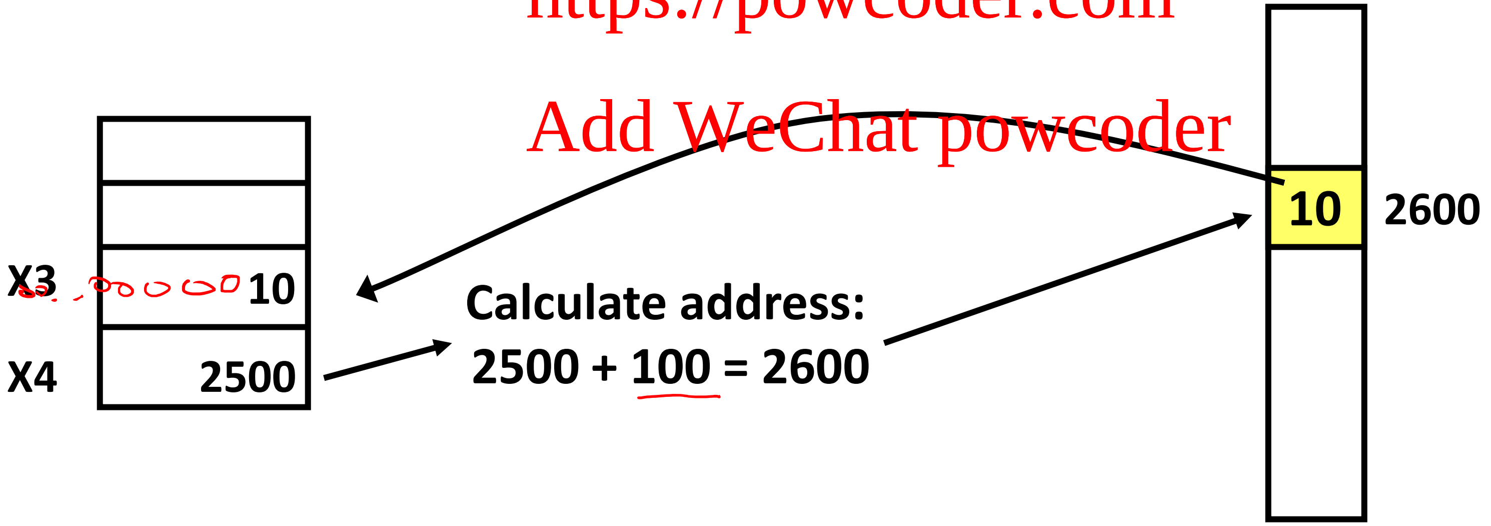
Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder



Load Instruction in Action

- `LDURB X3, [X4, #100]   // load byte`
  Retrieves 8-bit value from memory location (X4+100) and puts the result into X3. The other 56 most significant bits are 0—zero extended

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder



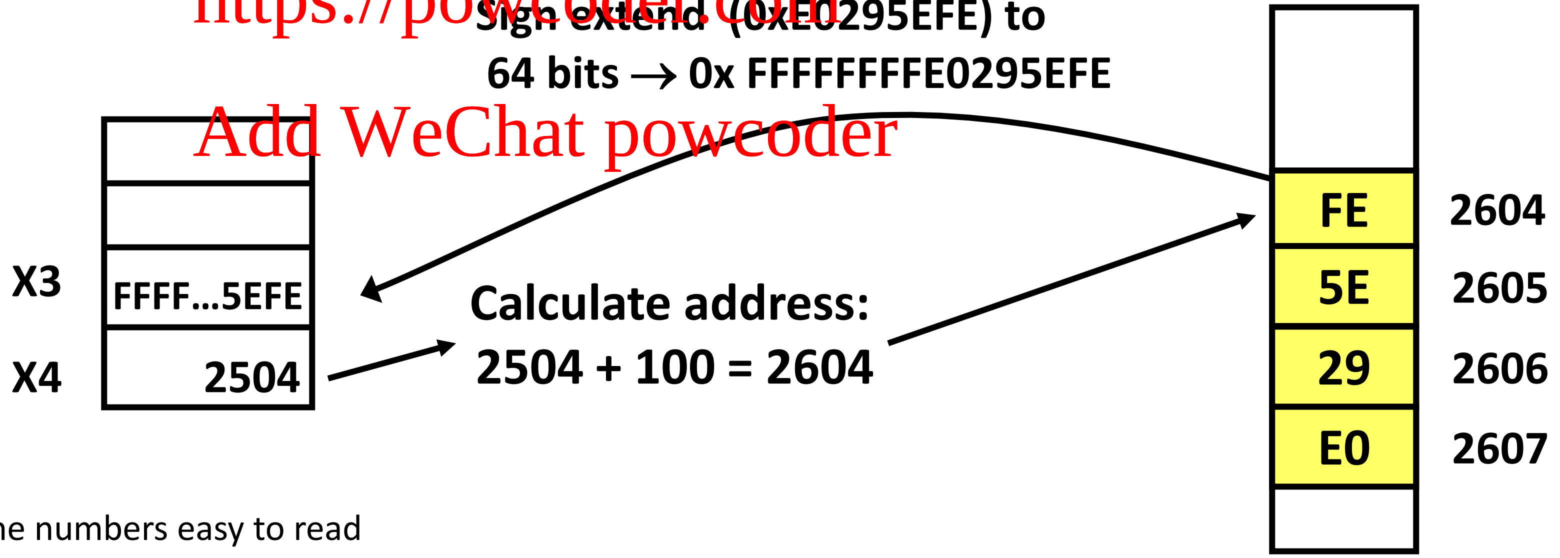
Load Instruction in Action – Example #2

- `LDURSW X3, [X4, #100]  // load signed word`
  Retrieves 4-byte value from memory location (X4+100) and puts the result into X3 (sign extended)

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder



Example mixes decimal and hex to keep the numbers easy to read
Recall that E = `1110` and thus is treated as a negative 2's complement number

Load Instruction in Action – Example #2

ARM ISA

- `LDURSW X3, [X4, #100]  // load signed word`
  Retrieves 4-byte value from memory location (X4+100) and puts the result into X3 (sign extended)

Assignment Project Exam Help

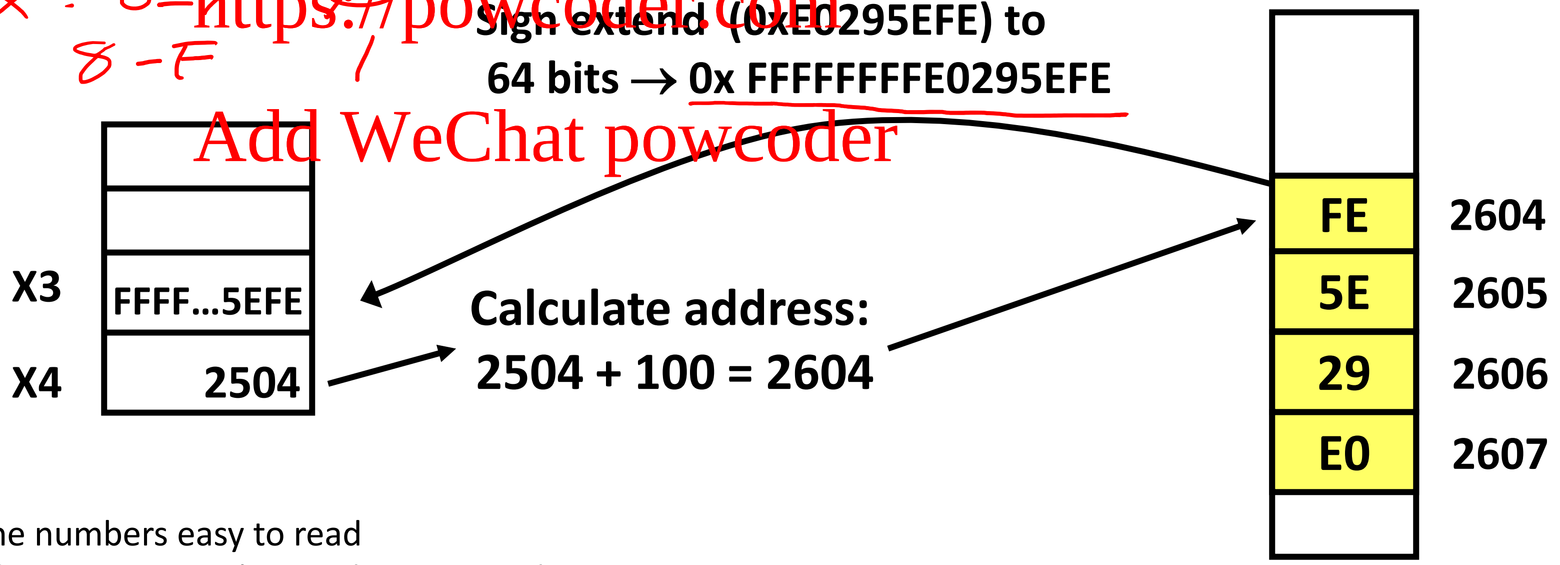
https://powcoder.com

Add WeChat powcoder

E = 14 HEX: 0-7 / 8-F

1 1 1 1 1 1 0

_ E 0 2 9 5 E F E

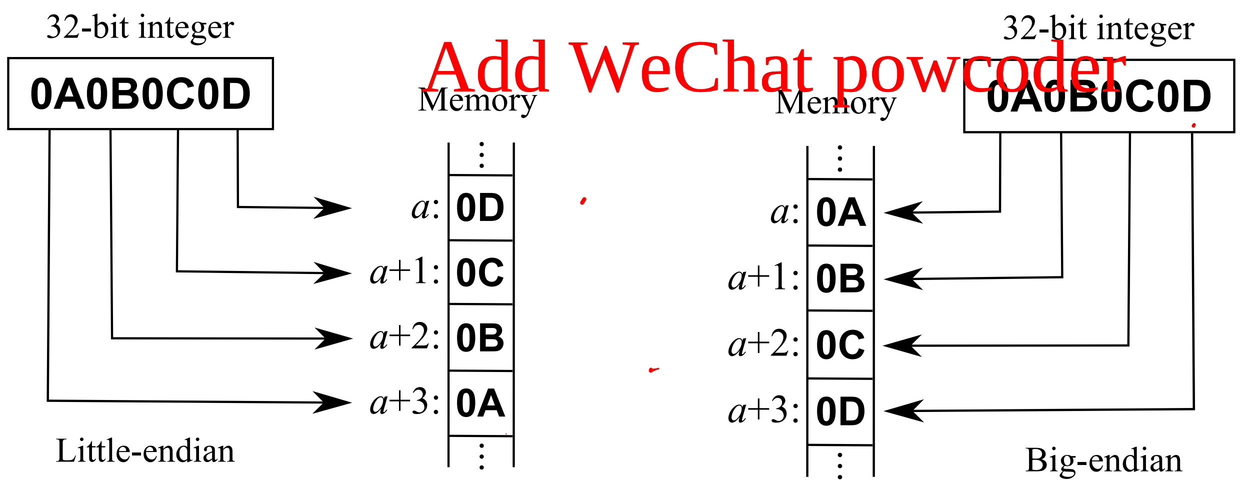


Example mixes decimal and hex to keep the numbers easy to read
Recall that E = `1110` and thus is treated as a negative 2's complement number

Big Endian vs. Little Endian

- Endian-ness: ordering of bytes within a word
  - Little - increasing numeric significance with increasing memory addresses
  - Big – The opposite, most significant byte first
  - The Internet is big endian, x86 is little endian, LEG and ARMv8 can switch
    - But in general assume little endian. (Figures from Wikipedia)

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder



Logistics

- There are 4 videos for lecture 4
  - L4_1 – ARM-ISA_Arithmetic-Logical_Instructions
  - L4_2 – ARM-ISA_Memory-Instructions
  - L4_3 – ARM-ISA_Memory-Instructions_Examples
  - L4_4 – C-to-Assembly
- There are two worksheets for lecture 4
  1. LEGv8 Assembly
  2. C to Assembly

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

L4_3 ARM-ISA_Memory-Instructions_Examples

Assignment Project Exam Help

https://powcoder.com

EECS 370 – Introduction to Computer Organization – Fall 2020

Add WeChat powcoder

Learning Objectives

- Recognize the set of instructions for ARM Architecture (ISA) and be able to describe the operations and operands for instructions
  - LEGv8 subset
- Ability to create simple ARM assembly programs, e.g., using mathematical, logic, and memory operations

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

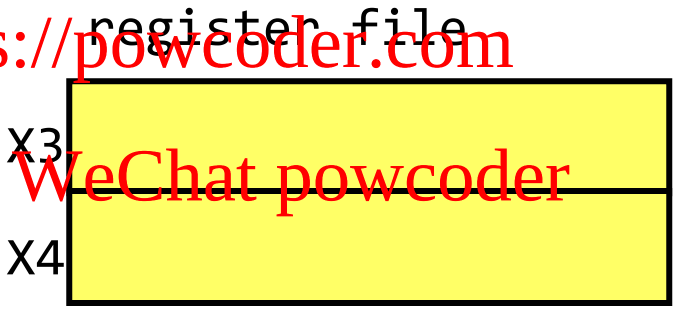
Example Code Sequence #1

- What is the final state of memory once you execute the following instruction sequence? (assume X5 has the value of 0)

ARM LEGv8

```
LDUR     X4, [X5, #100]
LDURB    X3, [X5, #102]
STUR     X3, [X5, #100]
STURB    X4, [X5, #102]
```

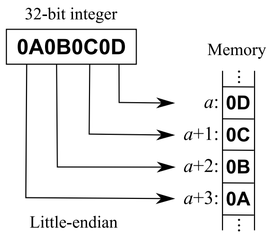
Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder



Memory
(each location is 1 byte)

workspace	start	
	0x02	100
	0x03	101
	0xFF	102
	0x05	103
	0xC2	104
	0x06	105
	0xFF	106
	0xE5	107

little endian



ARM ISA

Example Code Sequence #1

- What is the final state of memory once you execute the following instruction sequence? (assume X5 has the value of 0)

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

ARM LEGv8

```
~~1~~ LDUR     X4, [X5, #100]
~~2~~ LDURB    X3, [X5, #102]
~~3~~ STUR     X3, [X5, #100]
4     STURB    X4, [X5, #102]
```

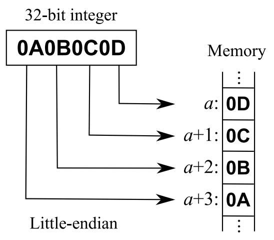
register file

3 2 ~~1~~	X3	00000000000000FF
1 ~~1~~	X4	E5 FF06C205FF03$\boxed{02}$

Memory
(each location is 1 byte)

workspace	start		
FF	0x02	100	~~1~~ ~~1~~
00	0x03	101	
02	0xFF	102	~~4~~ ~~4~~
00	0x05	103	
00	0xC2	104	
00	0x06	105	
00	0xFF	106	
00	0xE5	107	

little endian



Example Code Sequence #1

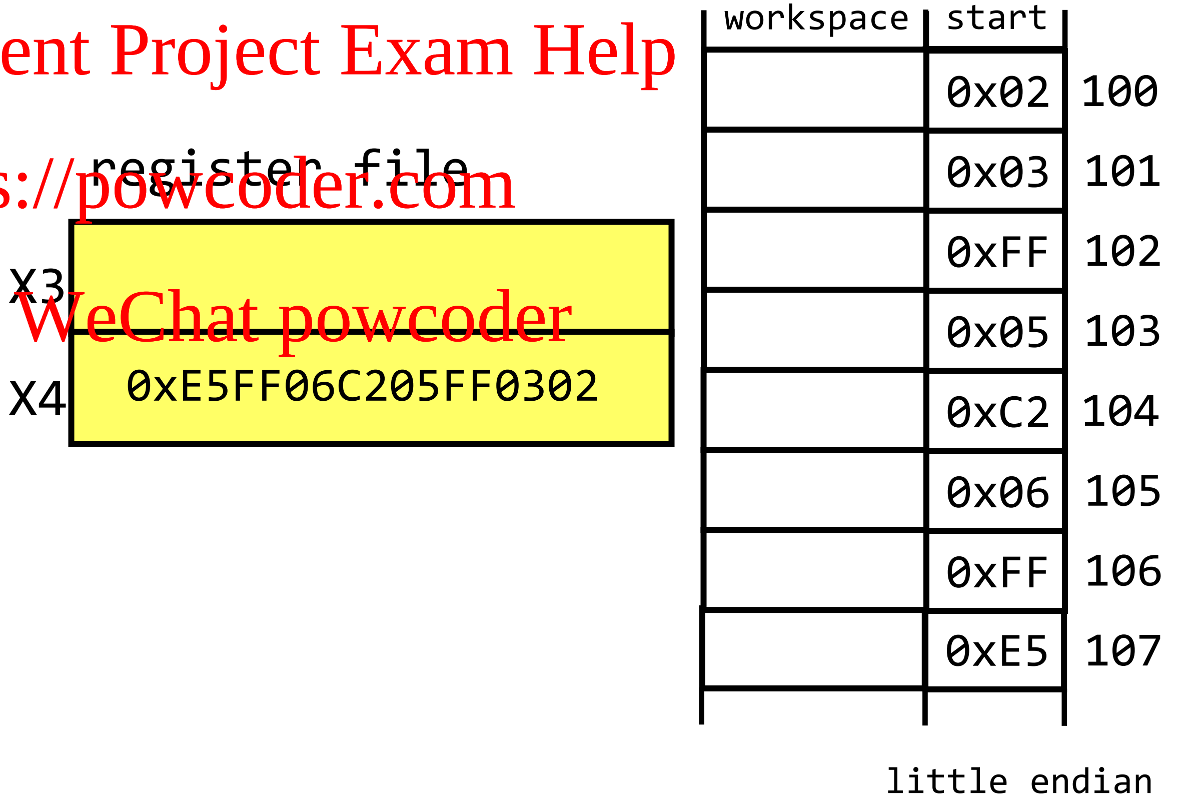
ARM ISA

- What is the final state of memory once you execute the following instruction sequence? (assume X5 has the value of 0)

ARM LEGv8

```
LDUR    X4, [X5, #100]
LDURB   X3, [X5, #102]
STUR    X3, [X5, #100]
STURB   X4, [X5, #102]
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder



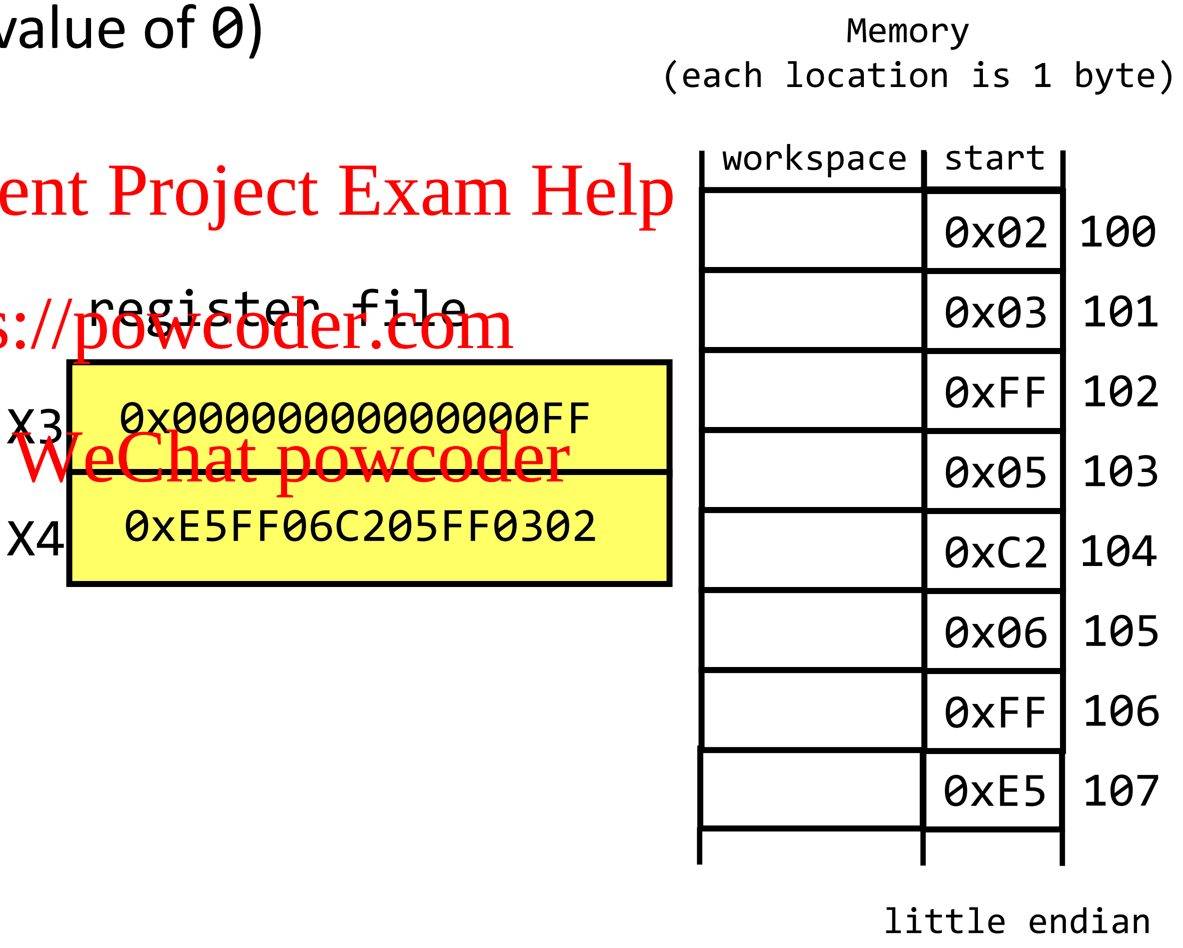
Example Code Sequence #1

- What is the final state of memory once you execute the following instruction sequence? (assume X5 has the value of 0)

ARM LEGv8

```
LDUR     X4, [X5, #100]
LDURB    X3, [X5, #102]
STUR     X3, [X5, #100]
STURB    X4, [X5, #102]
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder



ARM ISA

Example Code Sequence #1

- What is the final state of memory once you execute the following instruction sequence? (assume X5 has the value of 0)

ARM LEGv8

```
LDUR     X4, [X5, #100]
LDURB    X3, [X5, #102]
STUR     X3, [X5, #100]
STURB    X4, [X5, #102]
```

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

register file

X3	0x00000000000000FF
X4	0xE5FF06C205FF0302

Memory
(each location is 1 byte)

workspace	start	
0xFF	0x02	100
0x00	0x03	101
0x00	0xFF	102
0x00	0x05	103
0x00	0xC2	104
0x00	0x06	105
0x00	0xFF	106
0x00	0xE5	107

little endian

ARM ISA

Example Code Sequence #1

- What is the final state of memory once you execute the following instruction sequence? (assume X5 has the value of 0)

ARM LEGv8

```
LDUR    X4, [X5, #100]
LDURB   X3, [X5, #102]
STUR    X3, [X5, #100]
STURB   X4, [X5, #102]
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

register file

X3	0x00000000000000FF
X4	0xE5FF06C205FF0302

Memory
(each location is 1 byte)

workspace	start	
0xFF	0x02	100
0x00	0x03	101
0x02	0xFF	102
0x00	0x05	103
0x00	0xC2	104
0x00	0x06	105
0x00	0xFF	106
0x00	0xE5	107

little endian

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

Pause

The next example is a review of Lecture 4 worksheet 1.

Pause, complete the worksheet, then proceed.

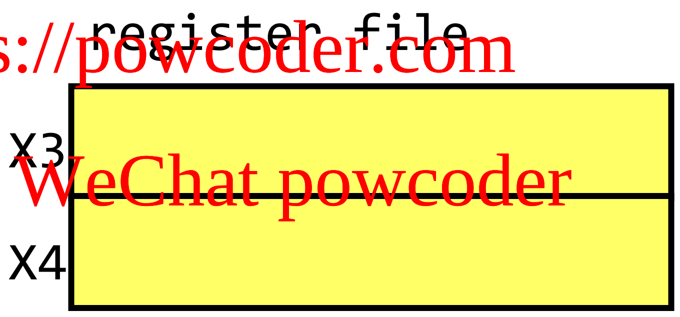
Example Code Sequence #2

- What is the final state of memory once you execute the following instruction sequence? (assume X5 has the value of 0)

ARM LEGv8

```
LDUR     X4, [X5, #100]
LDURB    X3, [X5, #102]
STURB    X3, [X5, #103]
LDURSW   X4, [X5, #100]
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder



Memory
(each location is 1 byte)

workspace	start	
	0x02	100
	0x03	101
	0xFF	102
	0x05	103
	0xC2	104
	0x06	105
	0xFF	106
	0xE5	107

little endian

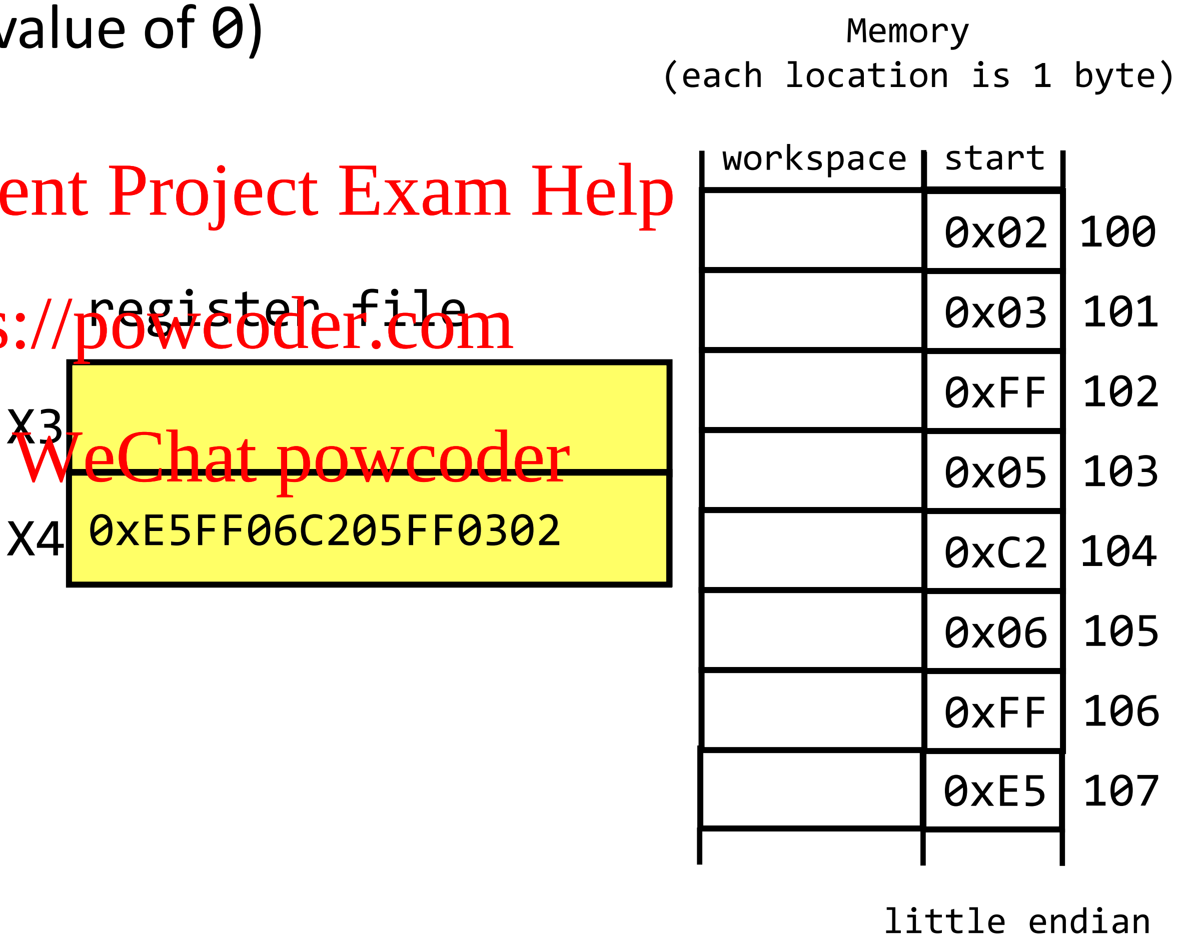
Example Code Sequence #2

- What is the final state of memory once you execute the following instruction sequence? (assume X5 has the value of 0)

ARM LEGv8

```
LDUR     X4, [X5, #100]
LDURB    X3, [X5, #102]
STURB    X3, [X5, #103]
LDURSW   X4, [X5, #100]
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder



Example Code Sequence #2

- What is the final state of memory once you execute the following instruction sequence? (assume X5 has the value of 0)

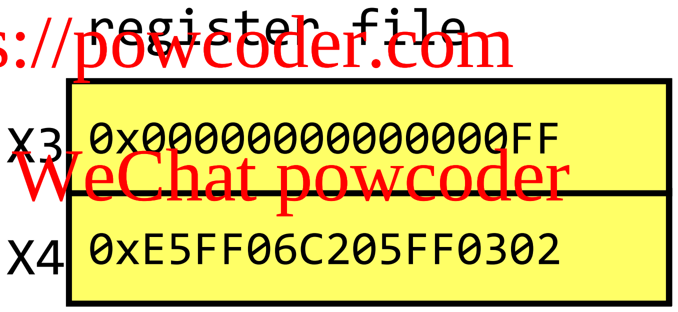
ARM LEGv8

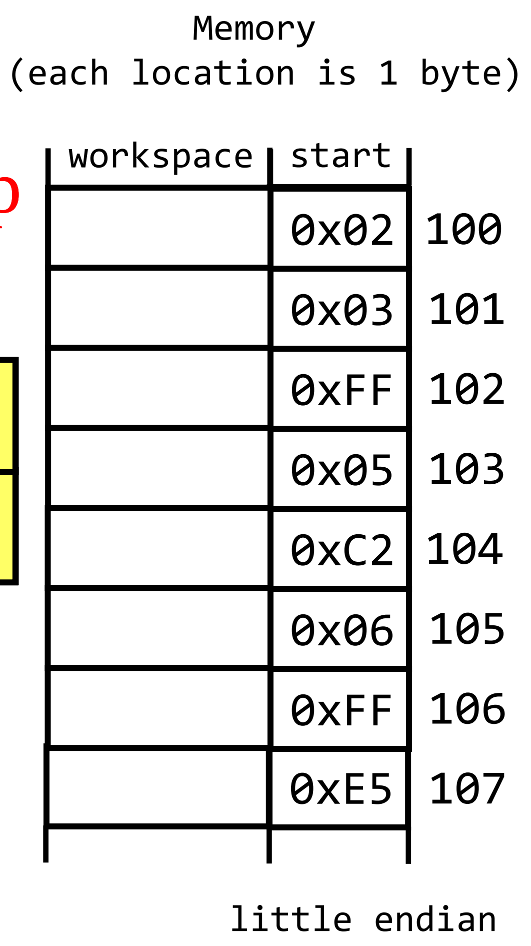
```
LDUR     X4, [X5, #100]
LDURB    X3, [X5, #102]
STURB    X3, [X5, #103]
LDURSW   X4, [X5, #100]
```

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder





workspace	start	
	0x02	100
	0x03	101
	0xFF	102
	0x05	103
	0xC2	104
	0x06	105
	0xFF	106
	0xE5	107

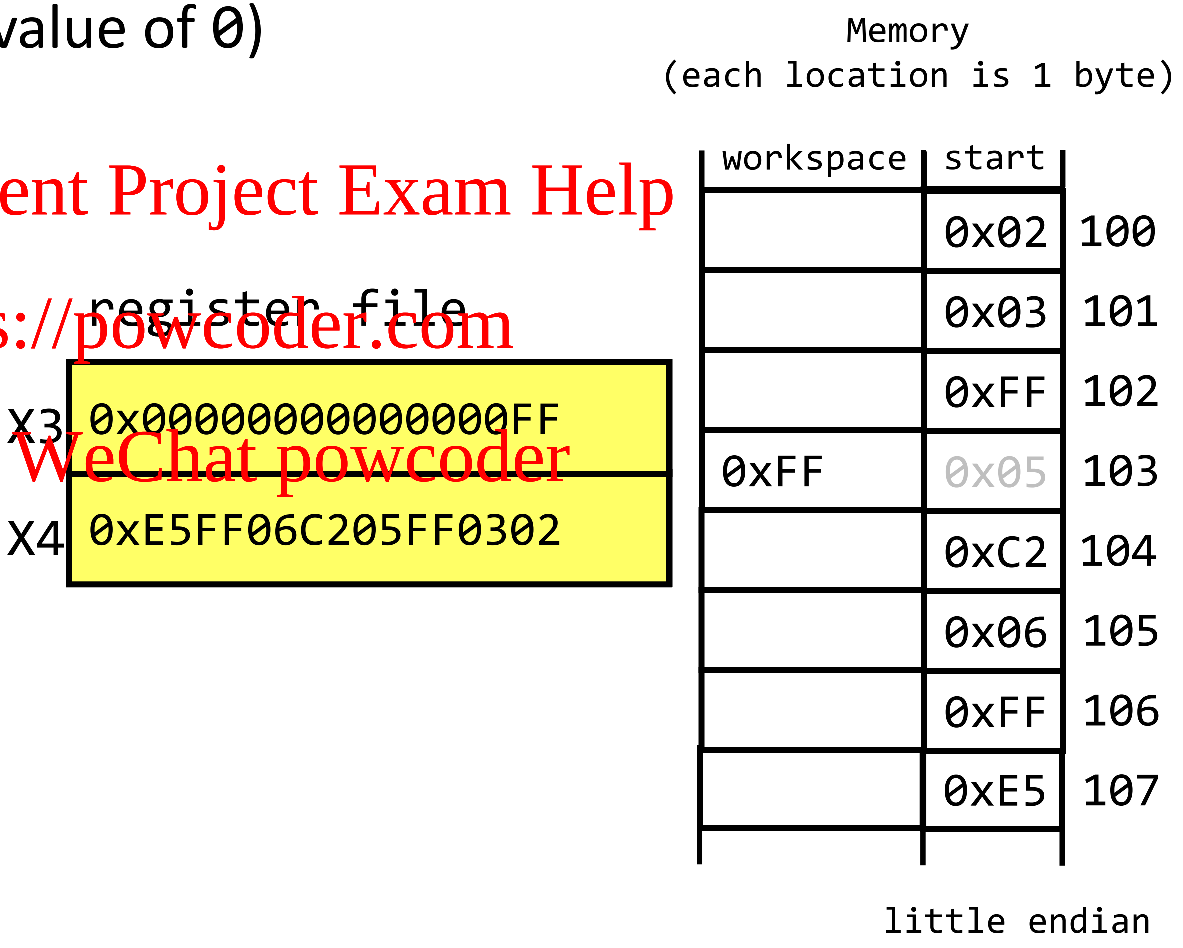
Example Code Sequence #2

- What is the final state of memory once you execute the following instruction sequence? (assume X5 has the value of 0)

ARM LEGv8

```
LDUR     X4, [X5, #100]
LDURB    X3, [X5, #102]
STURB    X3, [X5, #103]
LDURSW   X4, [X5, #100]
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder



ARM ISA

Example Code Sequence #2

- What is the final state of memory once you execute the following instruction sequence? (assume X5 has the value of 0)

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

ARM LEGv8

```
LDUR    X4, [X5, #100]
LDURB   X3, [X5, #102]
STURB   X3, [X5, #103]
LDURSW  X4, [X5, #100]
```

register file

X3	0x00000000000000FF
X4	0xFFFFFFFFFFFF0302

Memory
(each location is 1 byte)

workspace	start	
	0x02	100
	0x03	101
	0xFF	102
0xFF	0x05	103
	0xC2	104
	0x06	105
	0xFF	106
	0xE5	107

little endian

Logistics

- There are 4 videos for lecture 4
  - L4_1 – ARM-ISA_Arithmetic-Logical_Instructions
  - L4_2 – ARM-ISA_Memory-Instructions
  - L4_3 – ARM-ISA_Memory-Instructions_Examples
  - L4_4 – C-to-Assembly
- There are two worksheets for lecture 4
  1. LEGv8 Assembly
  2. C to Assembly

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

L4_4 C-to-Assembly

Assignment Project Exam Help

https://powcoder.com

EECS 370 – Introduction to Computer Organization – Fall 2020

Add WeChat powcoder

The material in this presentation cannot be copied in any form without written permission

Learning Objectives

- Translate C-code *statements* to ARM assembly code
  - Break down complex C-code instructions into a series of assembly operations
  - Map variables to registers

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

ARM ISA

Converting C to assembly – Example #1

Write ARM assembly code for the following C expression (the array holds 64-bit integers):

Register to variable mapping
`X1→a`
`X2→b`
`X3→i`
`X4→`*`start address of names`*

C-code instruction
`a = b + names[ i ];`

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

ARM ISA

Converting C to assembly – Example #1

Write ARM assembly code for the following C expression (the array holds 64-bit integers):

Register to variable mapping
X1→a
X2→b
X3→i
X4→*start address of names*

C-code instruction
`a = b + names[ i ];`

ARM LEGv8

```
LSL X5, X3, #3       // calculate array offset  i*8
ADD  X6, X4, X5      // calculate address of names[i]
LDUR X7, [X6, #0]    // load names[i]
ADD X1, X2, X7       // calculate b + names[i]
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Converting C to assembly – Example #2

ARM ISA

Lecture Worksheet #2

Write ARM assembly code for the following C expression (assume an `int` is 4 bytes, `unsigned char` is 1 byte)

Register to variable mapping	C-code instructions

X1→*pointer to y*	`struct { int a; unsigned char b, c; } y;`
	`y.a = y.b + y.c;`

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

ARM ISA

Lecture Worksheet #2

Converting C to assembly – Example #2

Write ARM assembly code for the following C expression (assume an `int` is 4 bytes, `unsigned char` is 1 byte)

Register to variable mapping
X1→*pointer to y*

C-code instructions

```
struct { int a; unsigned char b, c; } y;
y.a  = y.b + y.c;
```

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

ARM LEGv8

```
LDURB   X2,  [X1, #4]   // load y.b
LDURB   X3,  [X1, #5]   // load y.c
ADD     X4,  X2,  X3    // calculate y.b + y.c
STURW   X4,  [X1, #0]   // store y.a
```

See supplemental video for detailed explanation

How do you determine offsets for struct sub-fields?
Next lecture will detail

Logistics

- There are 4 videos for lecture 4
  - L4_1 – ARM-ISA_Arithmetic-Logical_Instructions
  - L4_2 – ARM-ISA_Memory-Instructions
  - L4_3 – ARM-ISA_Memory-Instructions_Examples
  - L4_4 – C-to-Assembly
- There are two worksheets for lecture 4
  1. LEGv8 Assembly
  2. C to Assembly

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder