Review Problem 2

❖ In assembly, set X0 to -X1.

SUB X0, X31, X1

5981 x0, x1, #18

SuBI xo, #19 X+

Basic Operations

(Note: just subset of all instructions)

Mathematic: ADD, SUB, MUL, SDIV

ADD X0, X1, X2
$$//$$
 X0 = X1+X2

Immediate (second input a constant) ADDI X0, X1, #100 // X0 = X1+100

$$X0, X1, #100 // X0 = X1+100$$

Logical: AND, ORR, EOR

Immediate

$$//X0 = X1&X2$$

ANDI X0, X1, #7 // X0 = X1&b0111

Shift: left & right logical (LSL, LSR)

LSL X0, X1, #4 // X1

$$// X0 = X1 << 4$$

Example: Take bits 6-4 of X0 and make them bits 2-0 of X1, zeros otherwise:

Memory Organization

"Byte addressing" means that the index points to a byte of memory. A memory address is an index into the array Viewed as a large, single-dimension array, with an address.

8 bits of data	<u> </u>
8 bits of data	<u> </u>
8 bits of data	-}>
8 bits of data	<u> </u>
8 bits of data	\sim
8 bits of data	}
8 bits of data	0

:

Memory Organization (cont.)

Bytes are nice, but most data items use larger units.

Double-word = 64 bits = 8 bytes

Word = 32 bits = 4 bytes

0	64 bits of data
000	64 bits of data
16	64 bits of data
24	64 bits of data

Registers hold 64 bits of data

2⁶⁴ bytes with byte addresses from 0 to 2⁶⁴-1

2⁶¹ double-words with byte addresses 0, 8, 16, ... 2⁶⁴-8

Double-words and words are aligned

i.e., what are the least 3 significant bits of a double-word address? 000 S

Addressing Objects: Endian and Alignment

journal h	000 001 010 101 111 111								Doublewo	
2720	215.28	223216	231224	239232	247240	255248	263.256		rd: 263.256	
		\		3					255.248 2	
• •	32 DWord 4		M DWord 3	16 DWord 2	& DWord 1	0 Dword 0			47240 239232	
/			Little Endian				\		Doubleword: 263256 255248 247240 239232 231224 223216 21528 2720	
\neq			9		0 0		9		15.28	
263.256	255.248	247240	239.232	$011 2^{31}.2^{24}$	010 $2^{23}.2^{16}$	215.28	27.20		27.20	

Big Endian: address of most significant byte = doubleword address Little Endian: address of least significant byte = doubleword address Motorola 68k, MIPS, IBM 360/370, Xilinx Microblaze, Sparc

ARM: can do either - this class assumes Little-Endian.

Intel x86, DEC Vax, Altera Nios II, Z80

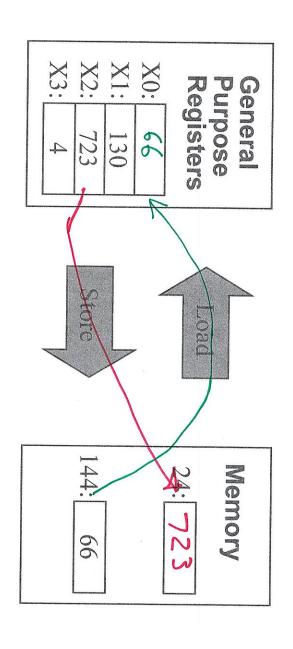
Data Storage

Pointer: Address (64 bits) Array: Sequence of locations _{0x1001} Integers: 64 bits (D-word) Characters: 8 bits (byte) char **(**) new new char[4]; int[10]; ASCII 71: 0x47 258; // 0x102 X:0x1008 0x100F 0x100D 0x100E 0x100C 0x100B 0x100A 0x1003 0x1007 0x10090x1005 0x1004 0x1002 0x1006 0x1000 | 0x47 0 000 0 000 0 0 000 0 7: 0x1018 0x101E 0x101D 0x101C 0x101A 0x1017 0x101F 0x101B 0x1019 0x1016 0x10130x1010 0x1015 0x1014 0x1011 0x1012 00 0 00 8 0 8 00 00 00 00 C 0x102F 0x102E 0x1029 0x102D 0x1028 0x1027 6x1020 0x102C 0x102A 0x1022 0x1021 0x102B 0x1026 0x1025 0x1024 0x1023 0

(Note: real compilers place local variables (the "stack") from beginng of memory, new'ed structures (the "heap") from end. We ignore that here for simplicity)

Loads & Stores

Loads & Stores move data between memory and registers All operations on registers, but too small to hold all data



Note: LDURB & STURB load & store bytes