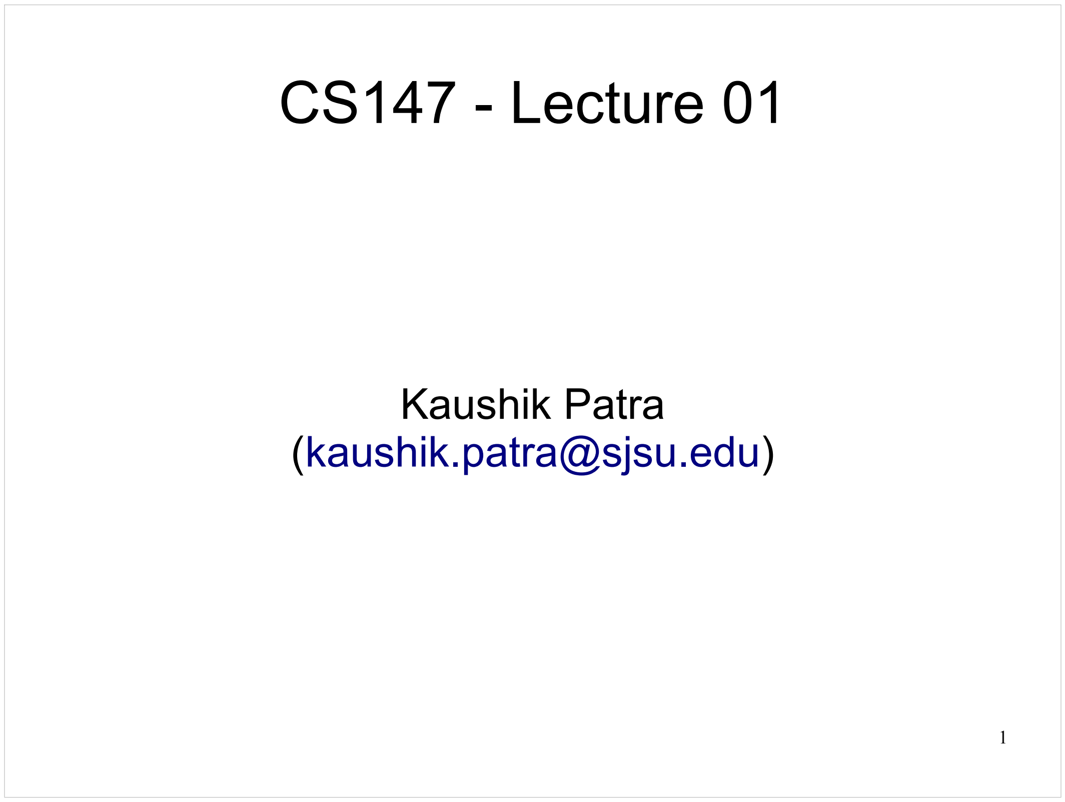
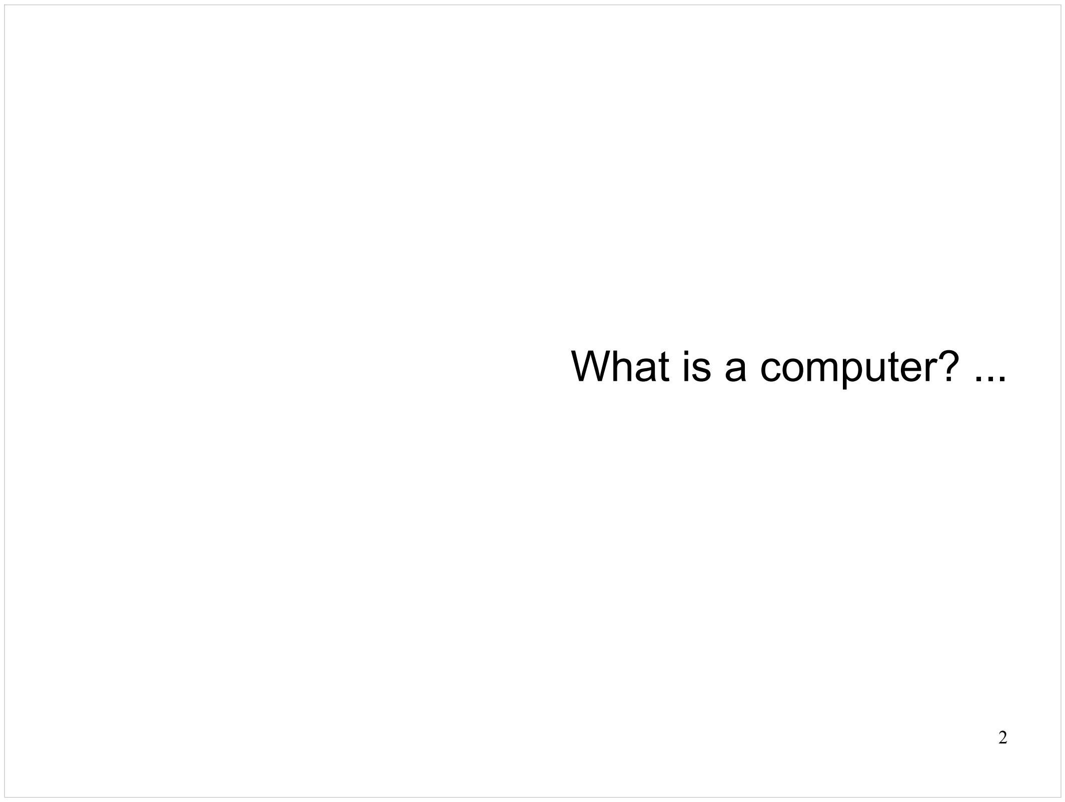


CS147 - Lecture 01

Kaushik Patra
(kaushik.patra@sjsu.edu)

1

- Topics
  - Introduction to computer
  - Basic of Instruction Set
  - Arithmetic & Logic Unit (ALU)



What is a computer? ...

Different 'avatar' of computer

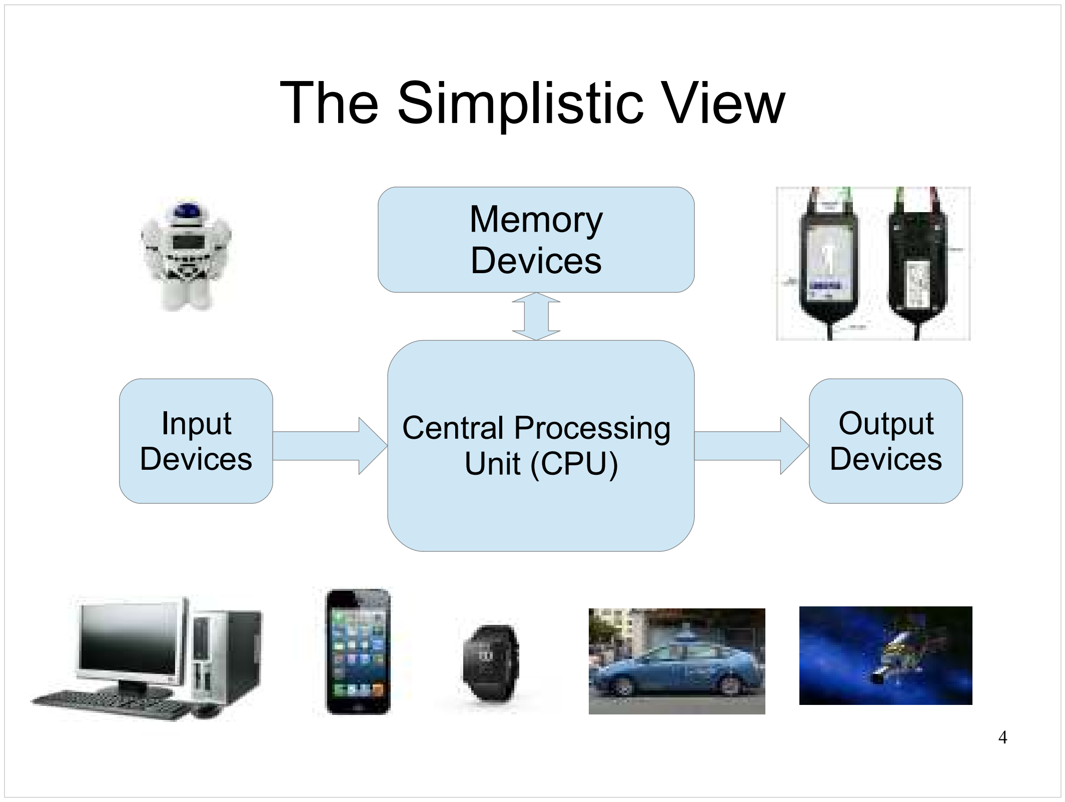
- Lexically 'computer' is 'what can compute.'

- In today's life computer is omnipresent – from personal gadgets to space exploration, from health related areas to entertainment.

- In early days desktops / PC were pretty much only representative of computer in common life.

- With advent of embedded technology, computer is now an intricate part of our personal as well as professional life.

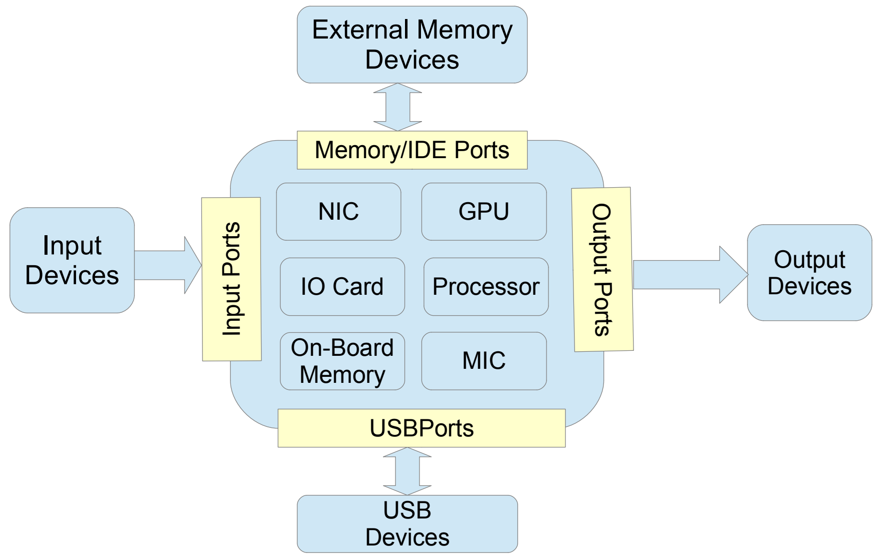
- From all these different forms and flavors of computers, how can we construct a common blue print which represent pretty much every one of them?



The Simplistic View

- All of the computer forms can be represented as a very basic diagram having following components.
  - Central Processing Unit (CPU) to precess incoming information.
  - Input Devices (keyboard, mouse, track pad, camera, microphone, sensors, etc.) to acquire incoming information.
  - Output devices (monitor, display, robot arms, printers, speakers, etc.) to manifest the outcome of computation.
  - Memory Devices (main memory, flash drive, hard disk, tape) to store and reuse information.
- Information flows from input devices (new) and memory (stored) into CPU.
- CPU Process the information and send it output devices for immediate use by users and memory for later use.
- Since involving memory storing latest information for later use, computers are state machines.

More into the CPU

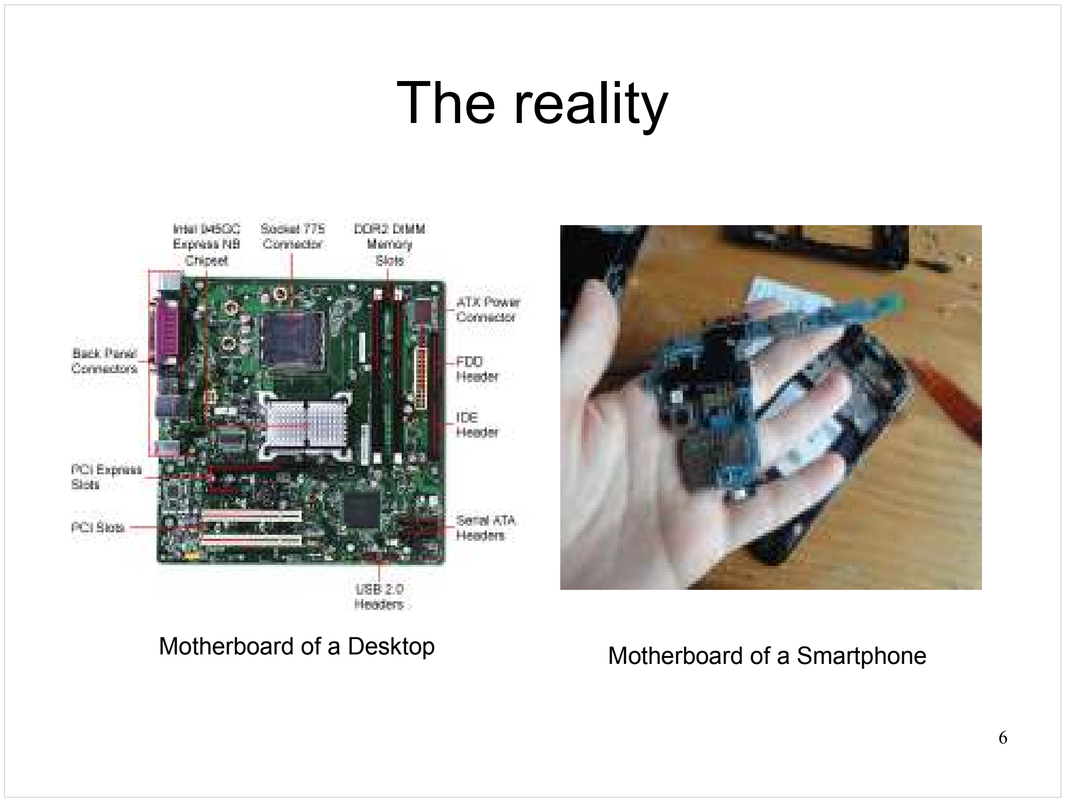


5

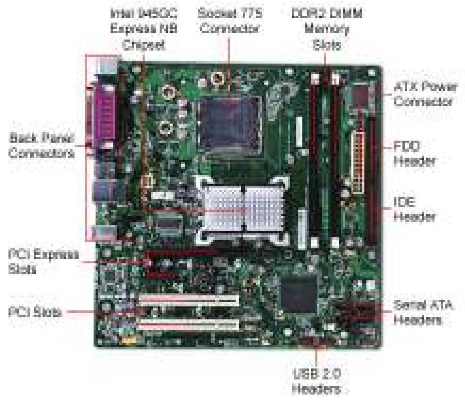
- The CPU may contain multiple parts like Processor (a.k.a. Micro-processor), GPU (Graphics Processing Units), NIC (Network Interface Card), IO Card (Input Output Card), MIC (Memory Interface Card), on board memory, and many more.

- All the external devices are connected to CPU using different types of ports.

- CS147 will concentrate study on the micro-processor, memory and their interaction. It'll also touch a little on the IO operations.



The reality



Motherboard of a Desktop

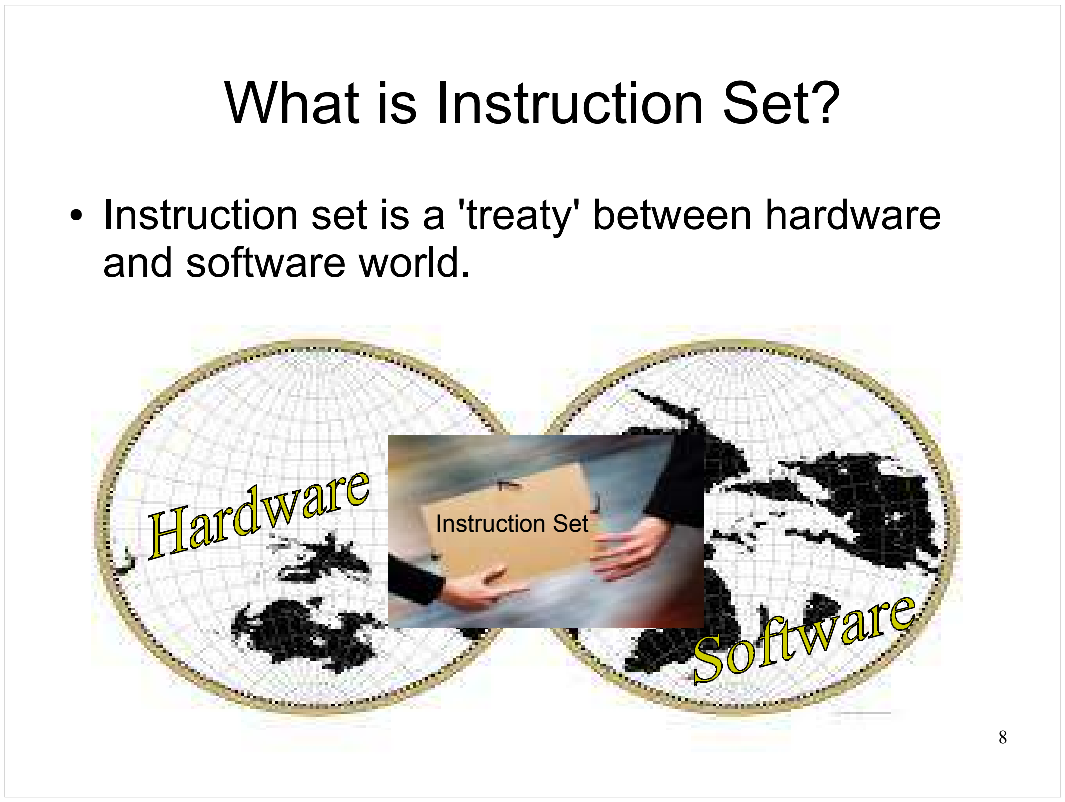
Motherboard of a Smartphone

6

- Placements of each individual components depends on the motherboard specification.

- With a smaller motherboard footprint requirement (e.g. smartphones) some of the components may be placed within single chip implementing SoC (System-on-Chip).

Basics of Instruction Set ...



What is Instruction Set?

- Instruction set is a 'treaty' between hardware and software world.

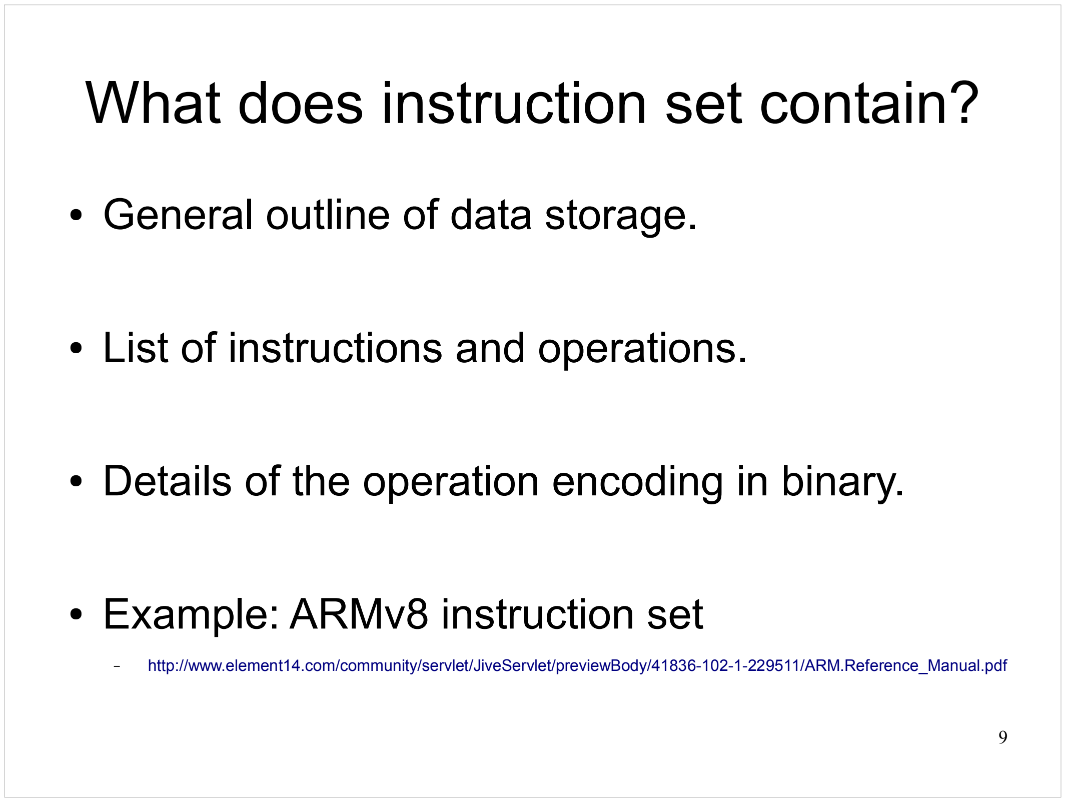
8

- It is common understanding between hardware and software world on the list of operations permitted on the target system.

- The list of operations acts as the specification for the hardware engineers on what to be implemented.

- The same list of operations acts as the specification for the software engineers using which the target program needs to be written.

- Before creating a new processor or extending an existing processor function architects from hardware and software field create together a new (or extension of) formal instruction set specification.



What does instruction set contain?

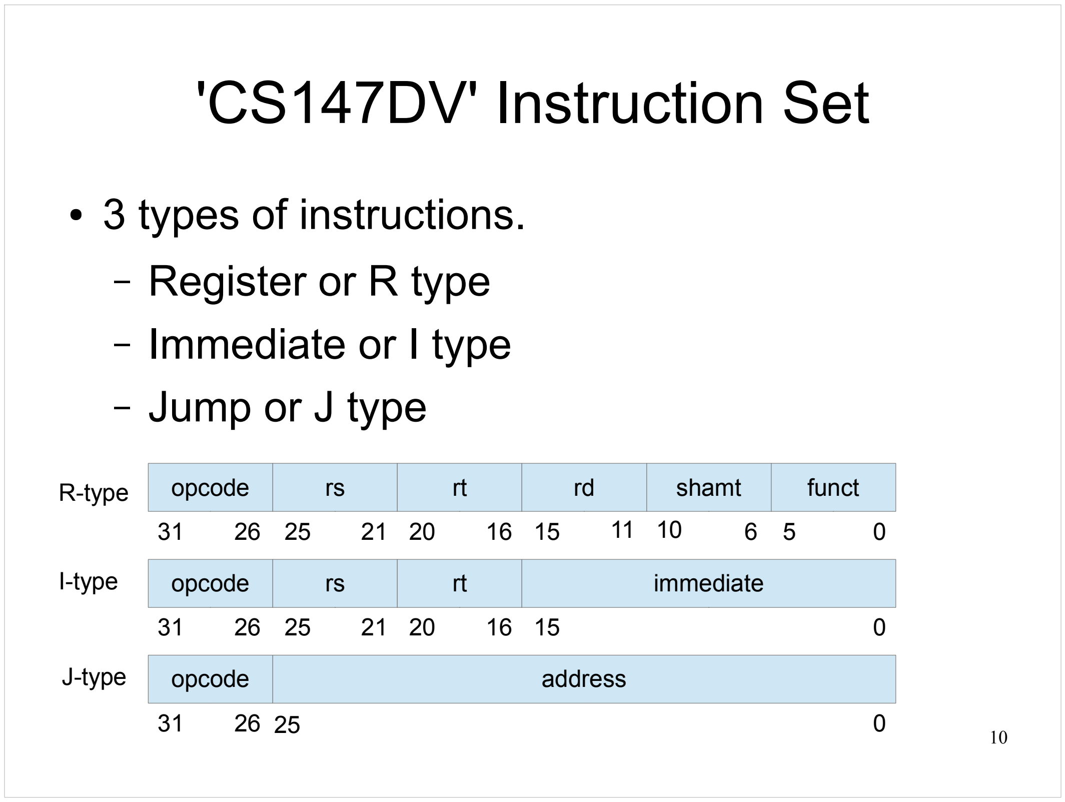
- General outline of data storage.
- List of instructions and operations.
- Details of the operation encoding in binary.
- Example: ARMv8 instruction set
  - http://www.element14.com/community/servlet/JiveServlet/previewBody/41836-102-1-229511/ARM.Reference_Manual.pdf

9

- Data storage may include:
  - number of bits to represent data (32 or 64 bits are common).
  - Addressable memory size or address space
  - Implicit memory access.
  - List of internal registers (data storage area with fastest access).
    - Explicit registers, which can be accessed explicitly.
    - Implicit registers, which are accessed implicitly.

- List of instruction and operation are created in mnemonic form for human perceptibility. e.g. 'add r1,r2' (r1 = r1 + r2). This is the lowest level human interpretable language used in computer programing, also known as assembly language. This part is mainly used by compilers to translate higher level language into architecture specific assembly language. This list establishes one to one correspondence to the underlying architecture.

- Computer understands binary numbers only. Thus it is required to translate each mnemonics in assembly language into machine interpretable binary numbers, or machine language. Instruction set also includes the translation guideline of mnemonic instruction into machine language.



'CS147DV' Instruction Set

- 3 types of instructions.
  - Register or R type
  - Immediate or I type
  - Jump or J type

R-type	opcode	rs	rt	rd	shamt	funct
	31 26	25 21	20 16	15 11	10 6	5 0
I-type	opcode	rs	rt	immediate		
	31 26	25 21	20 16	15 0		
J-type	opcode	address				
	31 26	25 0				

10

- OpCode is the common field of all type of instructions.
  - It is 6 bit long, hence max 64 instructions can be implemented.

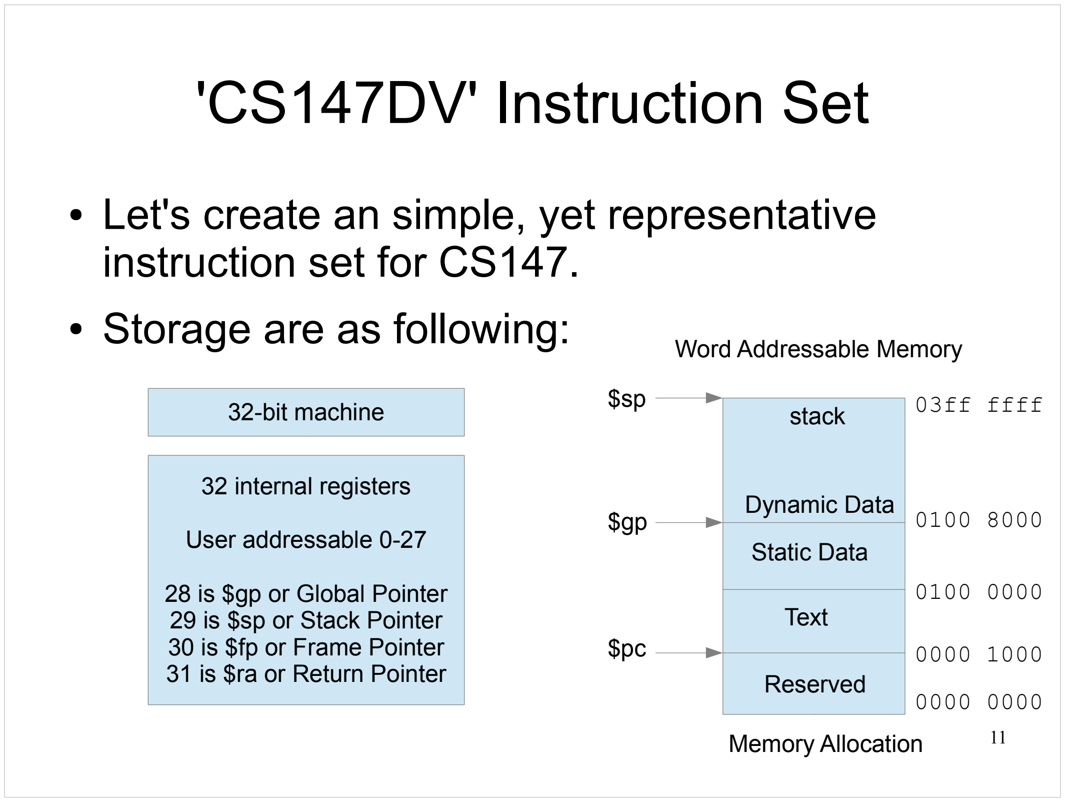
- Register or R type operations usually involves three registers and the target operation is done within content of those registers (rs, rt, and rd).

  - 'shamt' is the shift amount involved with the shift operation.
  - 'funct' is sub-operation (if needed) of a given 'opcode'.

- Immediate or I type operations involves two registers (one as source and another as destination register) and one immediate data.

  - Immediate numbers are sign extended or zero extended numbers. We'll see more explanation of sign extension in later sections.
  - Example: 4 bit to 8 bit sign extension – (**0**011 == **0000 0**011) , (**1**001 == **1111 1**001).
  - Example: 4 bit to 8 bit zero extension – (0011 == **0000** 0011) , (1001 == **0000** 1001).

- Jump or J type instruction takes 26-bit address.
  - The entire memory space of this architecture can be addressed by this 26-bit.



'CS147DV' Instruction Set

- Let's create an simple, yet representative instruction set for CS147.
- Storage are as following:

32-bit machine

32 internal registers

User addressable 0-27

28 is $gp or Global Pointer
29 is $sp or Stack Pointer
30 is $fp or Frame Pointer
31 is $ra or Return Pointer

Word Addressable Memory
$sp
stack
03ff ffff
$gp
Dynamic Data
0100 8000
Static Data
0100 0000
Text
$pc
0000 1000
Reserved
0000 0000
Memory Allocation

11

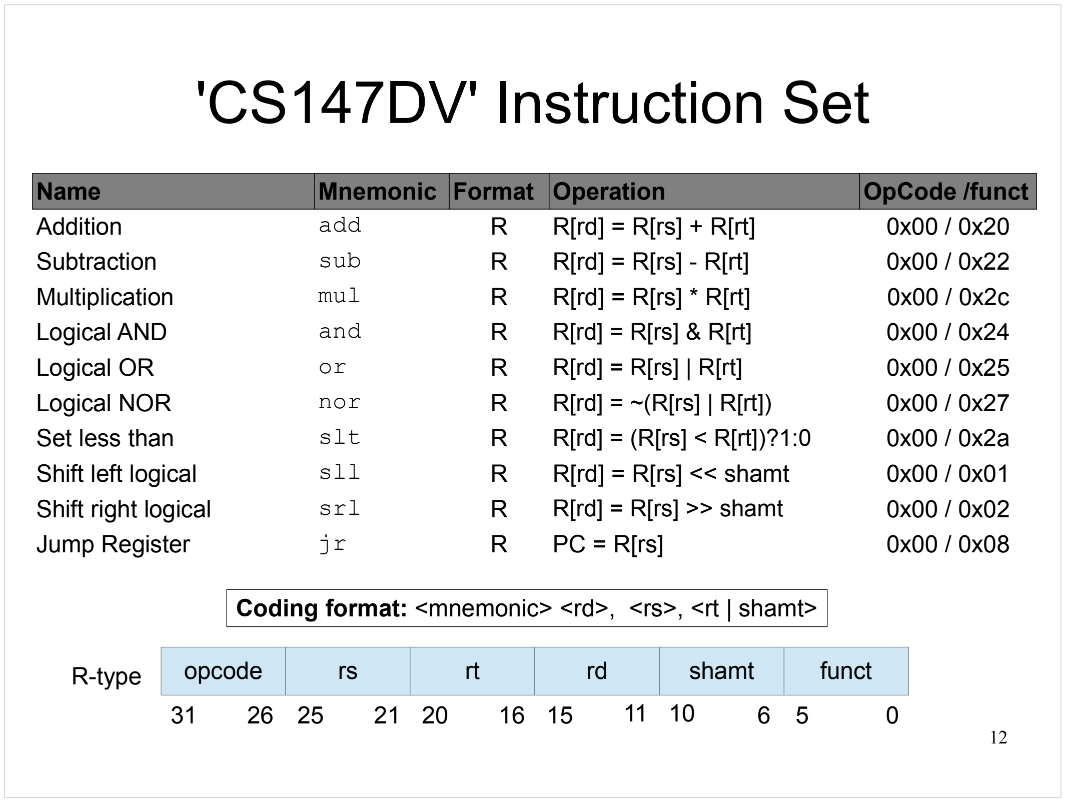
- It is a 32-bit architecture, i.e. all the operations are done with 32-bit precision, no longer than that.

- The memory is only word addressable. This means 32 bit needs to be read or written into a memory location (no half-word or byte granularity).

- There are 32 internal storage (32-bit registers) accessible with its index values from 0-31. e.g. r[0] is to access 1st register, r[10] is to access 11th register, etc.

  - Each of these registers are 32-bit.
  - 28 of them are user accessible (0-27).
  - r[28] is Global pointer or $gp.
  - r[29] is Stack pointer or $sp
  - r[30] is Frame pointer or $fp
  - r[31] is Return address pointer of $ra

- There is one special register $pc or Program Counter which determines the next instruction address. This register's power on address is '0x00001000', which means the target processor will start executing instruction at address '0x00001000' on power on.



'CS147DV' Instruction Set

Name	Mnemonic	Format	Operation	OpCode /funct
Addition	`add`	R	R[rd] = R[rs] + R[rt]	0x00 / 0x20
Subtraction	`sub`	R	R[rd] = R[rs] - R[rt]	0x00 / 0x22
Multiplication	`mul`	R	R[rd] = R[rs] * R[rt]	0x00 / 0x2c
Logical AND	`and`	R	R[rd] = R[rs] & R[rt]	0x00 / 0x24
Logical OR	`or`	R	R[rd] = R[rs] | R[rt]	0x00 / 0x25
Logical NOR	`nor`	R	R[rd] = ~(R[rs] | R[rt])	0x00 / 0x27
Set less than	`slt`	R	R[rd] = (R[rs] < R[rt])?1:0	0x00 / 0x2a
Shift left logical	`sll`	R	R[rd] = R[rs] << shamt	0x00 / 0x01
Shift right logical	`srl`	R	R[rd] = R[rs] >> shamt	0x00 / 0x02
Jump Register	`jr`	R	PC = R[rs]	0x00 / 0x08

Coding format: <mnemonic> <rd>, <rs>, <rt | shamt>

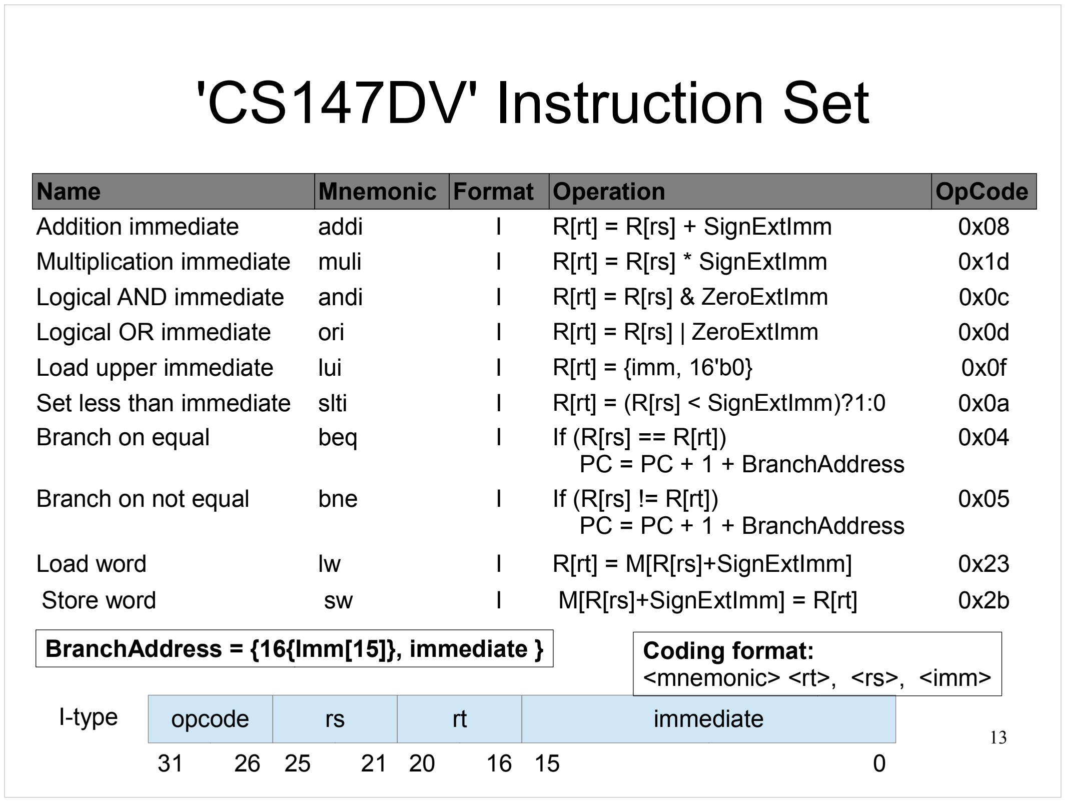
R-type	opcode	rs	rt	rd	shamt	funct
	31 26	25 21	20 16	15 11	10 6	5 0

12

- For example 'add r2, r1, r8' means 'r2 = r1 + r8'
- This will be represented in machine code as in binary:

opcode	000000
rs	00001
rt	01000
rd	00010
shamt	xxxxx (do not care)
funct	100000

- Putting it together would
  - **0000 00**00 001**0 1000** 0001 0**000 00**10 0000 in binary
  - 0x00281020 in hex
  - This is the fundamental job of assembler.



'CS147DV' Instruction Set

Name	Mnemonic	Format	Operation	OpCode
Addition immediate	addi	I	R[rt] = R[rs] + SignExtImm	0x08
Multiplication immediate	muli	I	R[rt] = R[rs] * SignExtImm	0x1d
Logical AND immediate	andi	I	R[rt] = R[rs] & ZeroExtImm	0x0c
Logical OR immediate	ori	I	R[rt] = R[rs] | ZeroExtImm	0x0d
Load upper immediate	lui	I	R[rt] = {imm, 16'b0}	0x0f
Set less than immediate	slti	I	R[rt] = (R[rs] < SignExtImm)?1:0	0x0a
Branch on equal	beq	I	If (R[rs] == R[rt]) PC = PC + 1 + BranchAddress	0x04
Branch on not equal	bne	I	If (R[rs] != R[rt]) PC = PC + 1 + BranchAddress	0x05
Load word	lw	I	R[rt] = M[R[rs]+SignExtImm]	0x23
Store word	sw	I	M[R[rs]+SignExtImm] = R[rt]	0x2b

BranchAddress = {16{Imm[15]}, immediate }

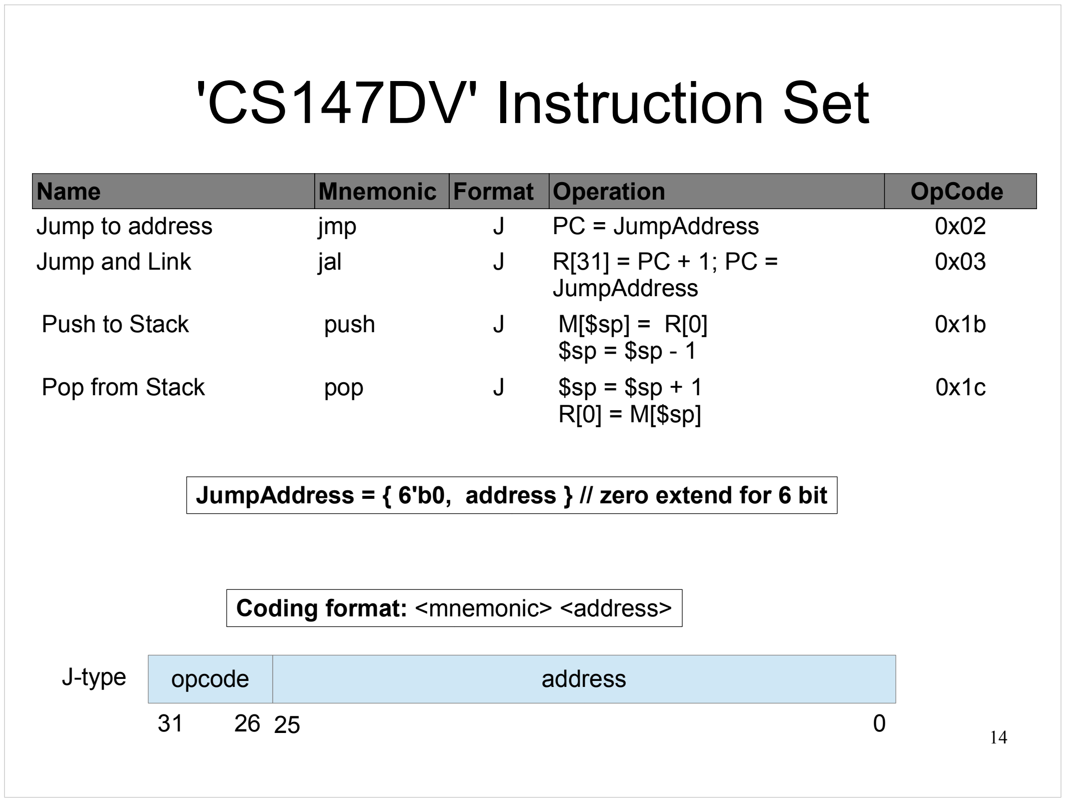
Coding format:
<mnemonic> <rt>, <rs>, <imm>

I-type	opcode	rs	rt	immediate
	31 26	25 21	20 16	15 0

13

Oprn \ Oprnd	0xc3b4	0x73b4
SignExtImm	0xffffc3b4	0x000073b4
ZeroExtImm	0x0000c3b4	0x000073b4

- For sign extension, look at the left most bit and repeat that for number of extension.
- For zero extension, pad zero at the lest for the required number of extension.
- If immediate is 0xc3b4 then branch address will be 0xffffc3b4.
- If immediate is 0x73b4 then the branch address will be 0x000073b4.

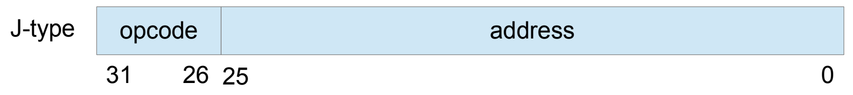


'CS147DV' Instruction Set

Name	Mnemonic	Format	Operation	OpCode
Jump to address	jmp	J	PC = JumpAddress	0x02
Jump and Link	jal	J	R[31] = PC + 1; PC = JumpAddress	0x03
Push to Stack	push	J	M[$sp] = R[0] $sp = $sp - 1	0x1b
Pop from Stack	pop	J	$sp = $sp + 1 R[0] = M[$sp]	0x1c

JumpAddress = { 6'b0, address } // zero extend for 6 bit

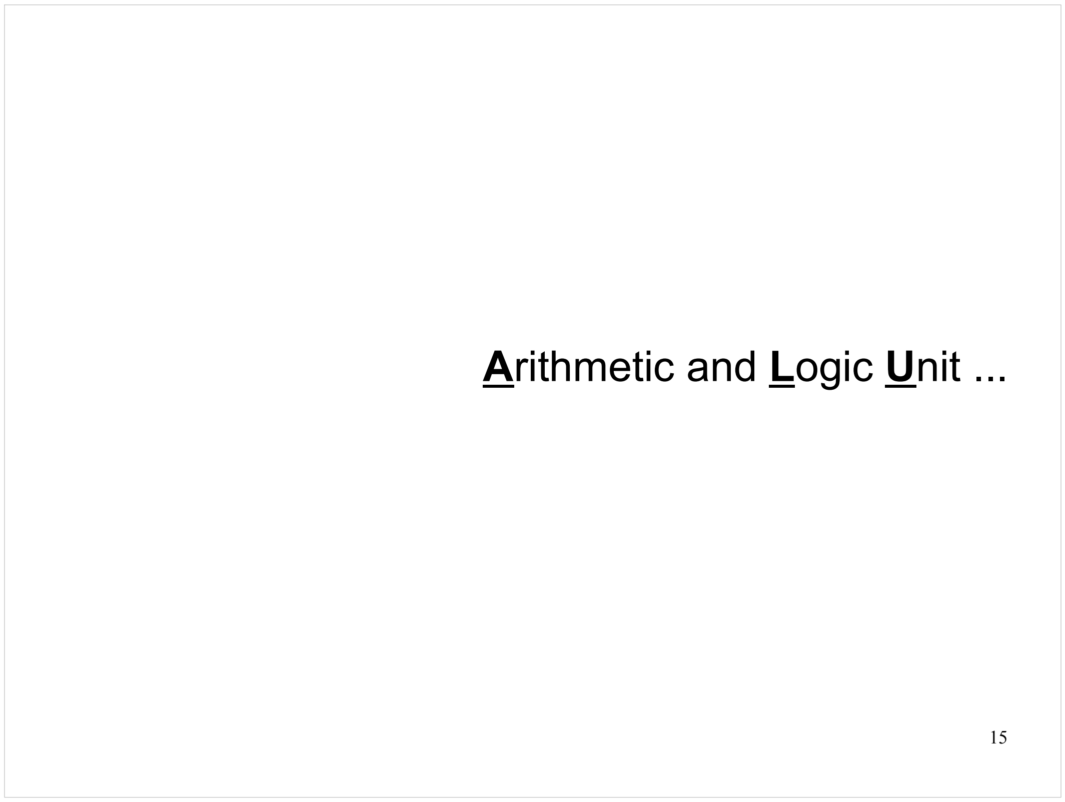
Coding format: <mnemonic> <address>



14

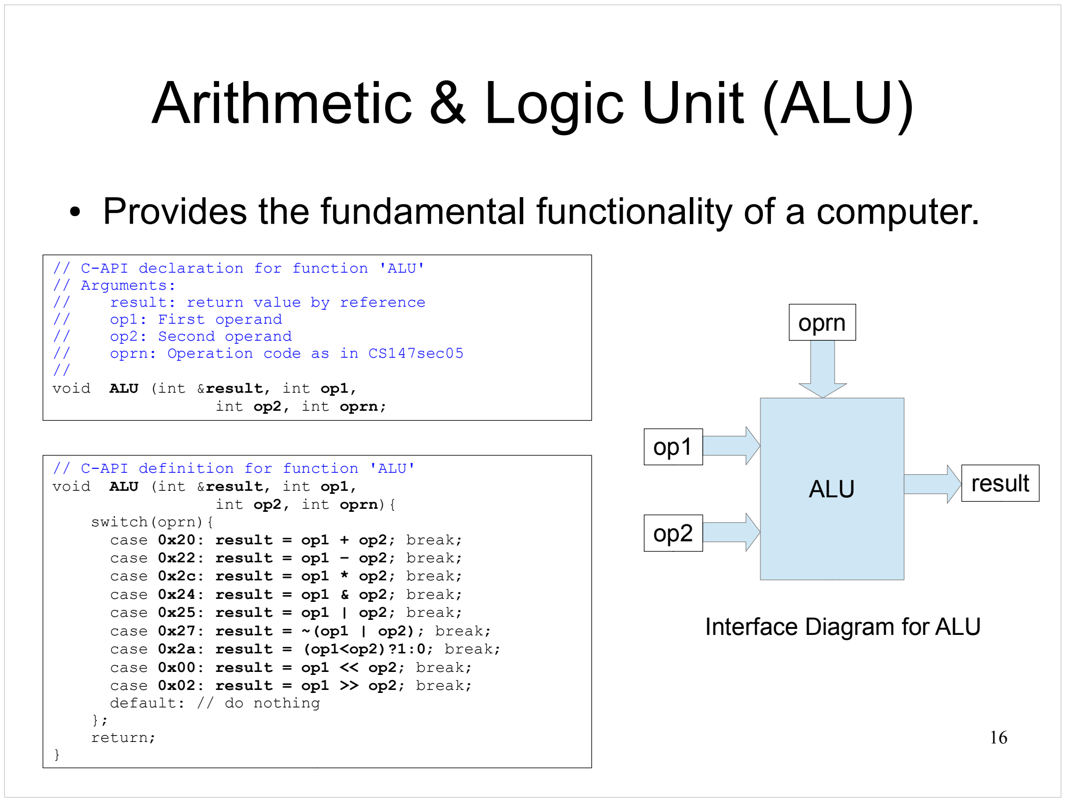
- Jump and link is very useful feature to implement return from sub-routines. Since it stores the return address at R[31], returning mechanism would involve call of 'jr R[31]'.

- The stack operation push / pop access R[0] implicitly.



Arithmetic and **L**ogic **U**nit ...

15

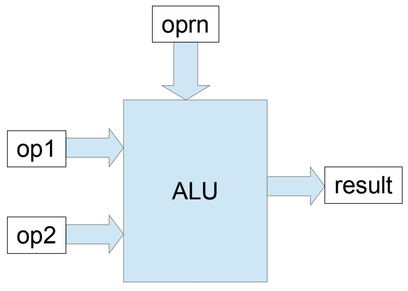


Arithmetic & Logic Unit (ALU)

- Provides the fundamental functionality of a computer.

```
// C-API declaration for function 'ALU'
// Arguments:
//    result: return value by reference
//    op1: First operand
//    op2: Second operand
//    oprn: Operation code as in CS147sec05
//
void  ALU (int &result, int op1,
                int op2, int oprn;
```

```
// C-API definition for function 'ALU'
void  ALU (int &result, int op1,
                int op2, int oprn){
    switch(oprn){
      case 0x20: result = op1 + op2; break;
      case 0x22: result = op1 - op2; break;
      case 0x2c: result = op1 * op2; break;
      case 0x24: result = op1 & op2; break;
      case 0x25: result = op1 | op2; break;
      case 0x27: result = ~(op1 | op2); break;
      case 0x2a: result = (op1<op2)?1:0; break;
      case 0x00: result = op1 << op2; break;
      case 0x02: result = op1 >> op2; break;
      default: // do nothing
    };
    return;
}
```

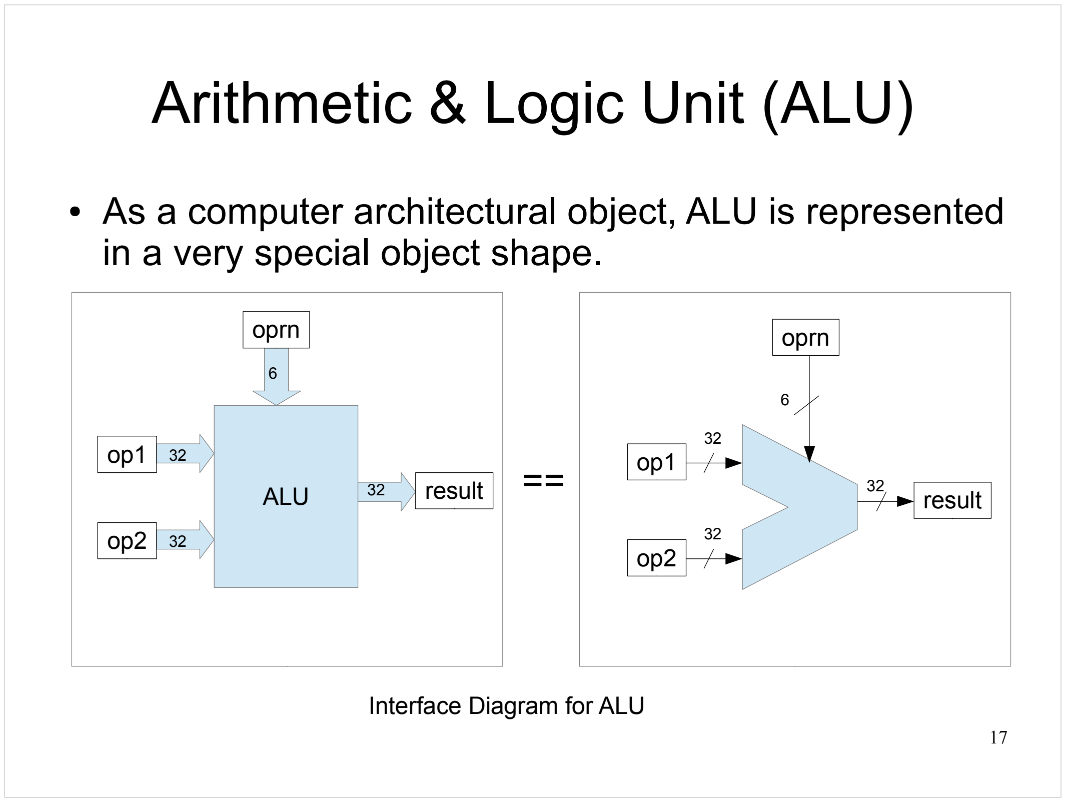


Interface Diagram for ALU

16

- ALU provides fundamental functionality of a computer. Any complex mathematical and logical program are broken down in terms two operand operations. For example: r = (a+b-c*d) is broken down in to following series of operations by compiler.

  - T1 = c * d
  - T2 = b – T1
  - r = a + T2



Arithmetic & Logic Unit (ALU)

- As a computer architectural object, ALU is represented in a very special object shape.

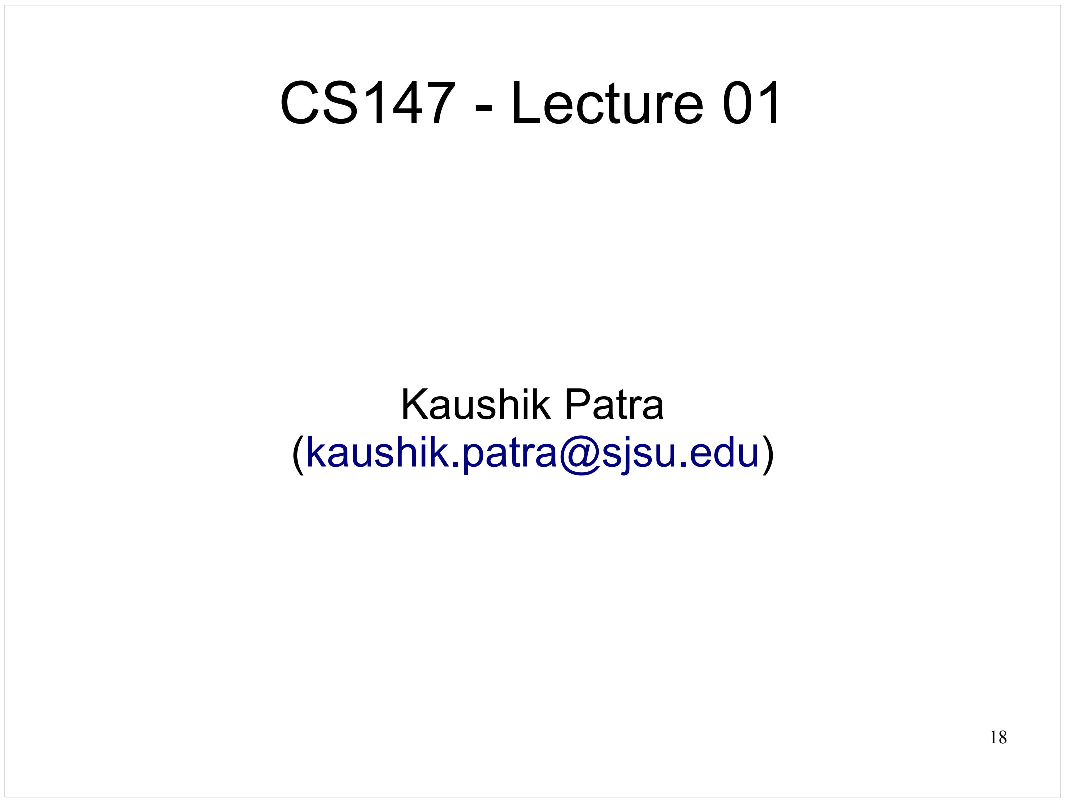
Interface Diagram for ALU

17

- Being a computer architectural object, it is necessary to include operation width. In our case it is 32 bit.

- Multiple bits are represented with single strike line indicating that the operations involves multiple bits. Plain line connection denotes single bit operation.

- The arrow indicates the direction of data flow – input, output or both ways.



CS147 - Lecture 01

Kaushik Patra
(kaushik.patra@sjsu.edu)

18

- Topics
  - Introduction to computer
  - Basic of Instruction Set
  - Arithmetic & Logic Unit (ALU)