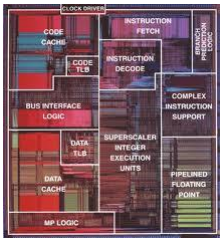
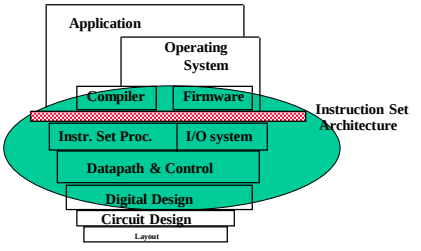
CS/SE 3340
Computer Architecture



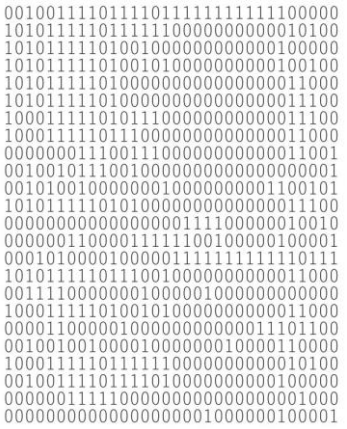


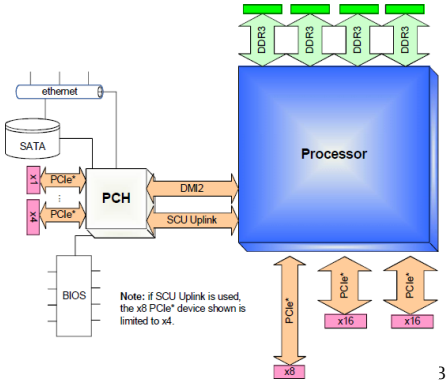
More MIPS Instructions

Based on slides from Prof. Aviral Shrivastava of ASU

Questions

- How to assemble MIPS assembly language statements to machine code?
- How does the processor access info. In memory?
- What is the functional view of the MIPS processor?
- How to access an array item with a constant/variable index?
- How to load/store a byte?





3

Programming Levels - Recap

High-level language program (e.g. C)

```
swap (int j, int k)
```

Assembly language program (MIPS)

```
swap:   sll   $2, $5, 2
        add   $2, $4, $2
        lw    $15, 0($2)
        lw    $16, 4($2)
        sw    $16, 0($2)
        sw    $15, 4($2)
        jr    $31
```

C - Compiler

Assembler

Machine (object) code (for MIPS)

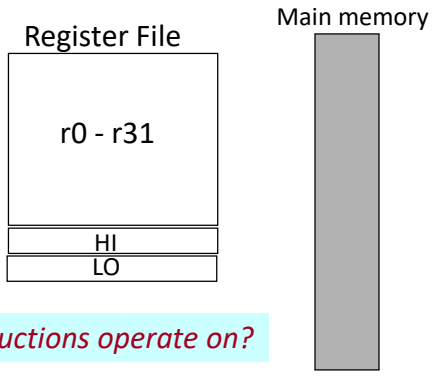
```
000000 00000 00101 00010000010000000
000000 00100 00010 00010000000100000
100011 00010 01111 00000000000000000
100011 00010 10000 00000000000000100
101011 00010 10000 00000000000000000
101011 00010 01111 00000000000000100
000000 11111 00000 00000000000001000
```

Where is the focus of Instruction Set Architecture (ISA) ?

4

MIPS Instruction Set Architecture

- Instruction Categories
  - Arithmetic
  - Data transfer (load/store)
  - Jump and Branch
  - Floating Point
    - coprocessor
  - Memory Management
  - Special



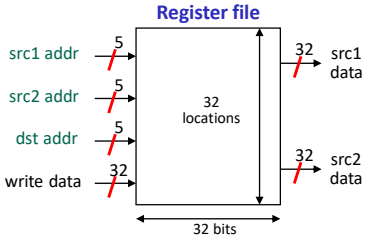
What do MIPS instructions operate on?

MIPS's three instruction formats:

op	rs	rt	rd	shamt	funct
op	rs	rt	immediate		
op	jump target				

MIPS Register File

- *All source operands* of *arithmetic instructions* must *be registers* from the MIPS register file
- *All destination operand* of *arithmetic instructions* must *be* written to a *register* of the MIPS register file
- Register file has thirty-two 32-bit registers
- In MIPS assembly language register names start with a '$'



MIPS Registers (recap*)

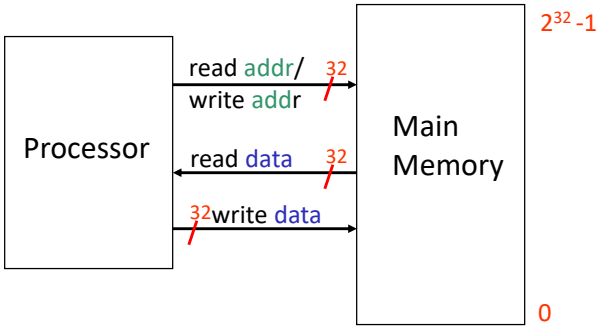
Register	Name	Usage
0	$zero	constant 0
1	$at	Reserved for assembler (pseudo-instructions)
2-3	$v0,$v1	Return function values
4-7	$a0-$a3	Function arguments
8-15 and 24-25	$t0-$t7, $t8,$t9	Temporaries (not preserved across call)
16-23	$s0-$s7	Save registers (preserved across call)
26-27	$k0,$k1	Reserved for kernel/OS
28	$gp	Pointer to global data area
29	$sp	Stack pointer. MARS initializes to 0x7FFF FFFC
30	$fp	Frame pointer
31	$ra	Return address, used by "link" instruction (HW)

**Slide 5 of session 03*

7

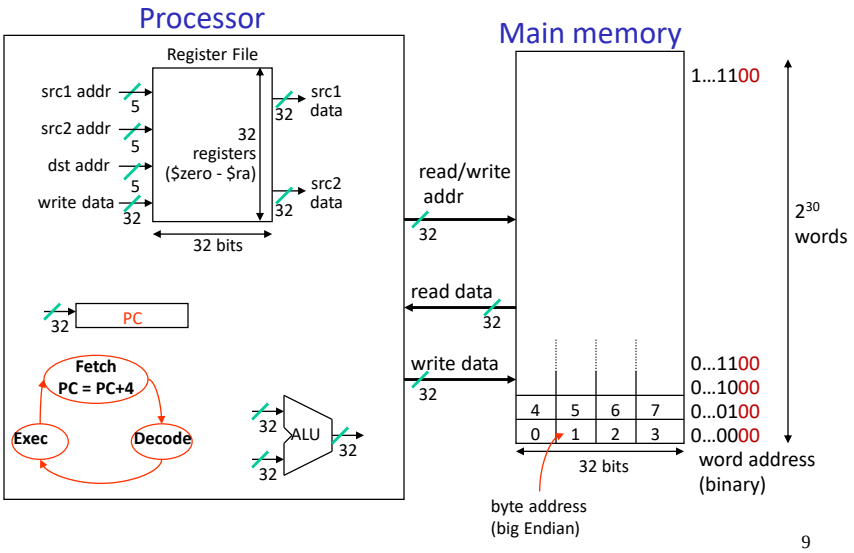
Processor and Main Memory

- Main memory is viewed as a single-dimension array of *bytes,* each of these has an ***address***
  - A *memory address* is an index into the array
- How to access *bytes* in main memory?



8

MIPS ISA Functional View



MIPS Arithmetic Instructions

- MIPS assembly language arithmetic instructions examples

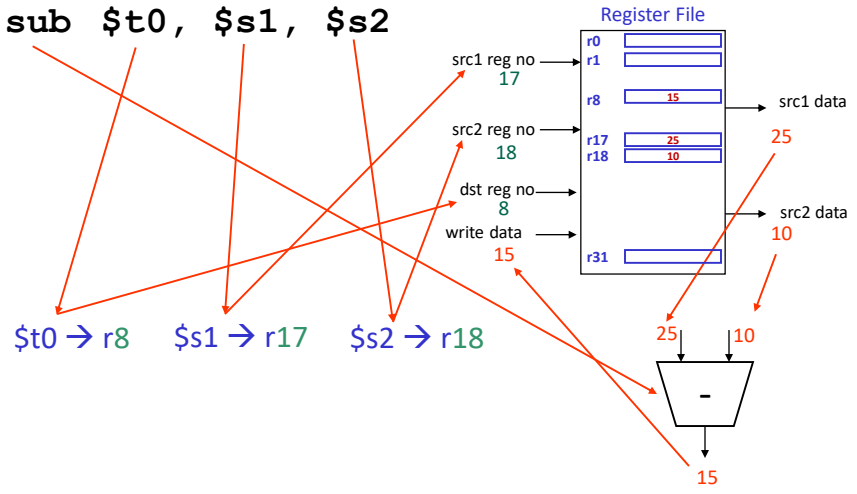
```
add  $t0, $s1, $s2
sub  $t0, $s1, $s2
```

what instruction format are these?

- Each arithmetic instruction performs only one operation
- Each arithmetic instruction specifies exactly three operands

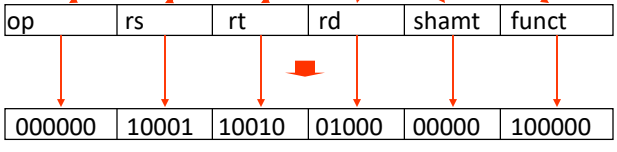
- All operands are registers!
  - `$t0, $s1,$s2 are in Register File`
- Order of operands is fixed

Arithmetic Instruction - Execution



Arithmetic Instructions - Assembly

- All MIPS instructions are 32 bits long
  - e.g., `add $t0, $s1, $s2`
  - Registers used: `$t0=$8, $s1=$17, $s2=$18`
- Instruction format:



Accessing Main Memory

- MIPS provides two basic *data transfer* instructions for accessing memory

```
lw    $t0, 4($s3)   #load word from memory
sw    $t0, 8($s3)   #store word to  memory
```

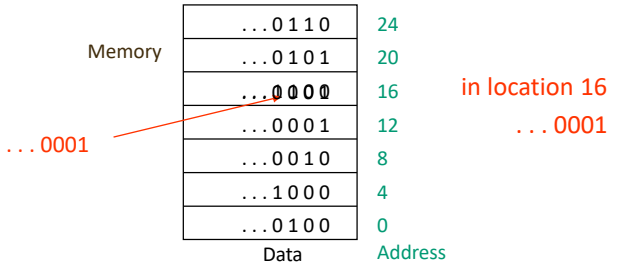
12 (above the 4) and 16 (below the 8)

- The data transfer instruction must specify
  - Memory address
  - Register source or destination
- The memory address is determined by
  - the content of the second register, and
  - the constant portion of the instruction
- Let's assume register $s3 contains 8 …

MIPS Memory Addressing

- The memory address is calculated by *summing the constant portion of the instruction and the contents of the second (base) register*
- Assuming $s3 contains the value 8

```
lw $t0, 4($s3) # what is loaded into $t0?
sw $t0, 8($s3) # where $t0's content is stored?
```

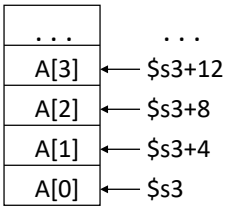


How to Access an Array?

- What is the MIPS assembly code for the following?

  `A[8] = A[2] – b`

- Assuming
  - Variable **`b`** is stored in **`$s2`**
  - **`A`** is an array of *words* and the base address of **`A`** is in **`$s3`**




```
lw   $t0, 8($s3)
sub  $t0, $t0, $s2
sw   $t0, 32($s3)
```

15

Assessing Array with a Variable Index

- What is the MIPS assembly code for the following?

  `c = A[i] - b`

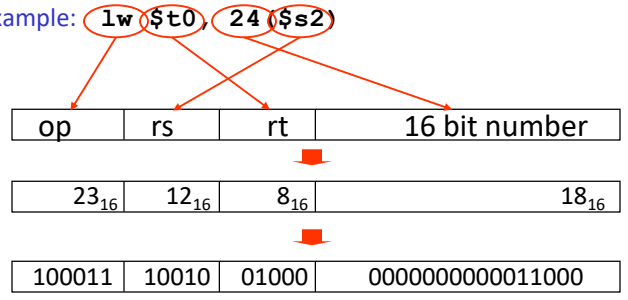
- Assuming
  - **`A`** is an array of n (e.g. 100) words
  - Base of **`A`** is in register **`$s4`**
  - Variables **`b`**, **`c`**, and **`i`** are in **`$s1`**, **`$s2`**, and **`$s3`**

```
add   $t1, $s3, $s3     #array index i is in $s3
add   $t1, $t1, $t1     #temp reg $t1 holds 4*i
add   $t1, $t1, $s4     #addr of A[i]
lw    $t0, 0($t1)
sub   $s2, $t0, $s1
```

16

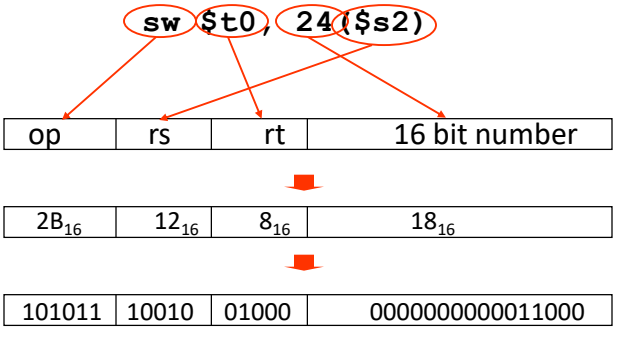
Assembly of Load-word Instruction

- Now let's look at the load-word and store-word instructions
  - What would the regularity principle dictate?
  - New principle: good design demands a compromise
- Introduce a new type of instruction format
  - I-type (Immediate)
  - (previous instruction format was R-type)
- Example: `lw $t0, 24($s2)`



I Format

Assembly of Store-word Instruction



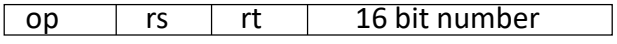
- A 16-bit number means access is limited to memory locations within a region of $\pm 2^{13}$ or 8,192 words ($\pm 2^{15}$ or 32,768 bytes) of the address in the base register $s2
- *How do we access to memory locations outside of this region?*

Loading and Storing Bytes

- MIPS provides special instructions to move bytes

```
lb $t0, 1($s3)   #load byte from memory
sb $t0, 6($s3)   #store byte to  memory
```

- Which byte in a word (4 bytes) that gets loaded and stored?
  - **lb** places the byte from memory in the rightmost 8 bits (LSB) of the destination register – *what happens to the other bytes?*
  - **sb** takes the byte from the rightmost 8 bits of a register and writes it to a byte in memory

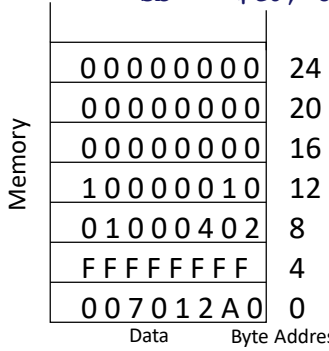


Example of Load and Store Bytes

- Given following code sequence and memory state (contents are given in hexadecimal and the processor use big Endian format), what is the state of the memory after executing the code?

```
add   $s3, $zero, $zero
lb    $t0, 1($s3)
sb    $t0, 6($s3)
```

mem(4) = 0xFFFF70FF



- What value is left in $t0?

  $t0 = 0x00000070

- What if the machine was little Endian?

  mem(4) = 0xFF12FFFF
  $t0 = 0x00000012

MIPS Instructions Covered

Category	Instr	Op Code	Example	Meaning
Arithmetic (*R format*)	add	0 and 0x20	add $s1, $s2, $s3	$s1 = $s2 + $s3
	subtract	0 and 0x22	sub $s1, $s2, $s3	$s1 = $s2 - $s3
Data transfer (*I format*)	load word load byte	0x23 0x20	lw $s1, 100($s2)	$s1 = Memory($s2+100)
	store word store byte	0x2B 0x28	sw $s1, 100($s2)	Memory($s2+100) = $s1