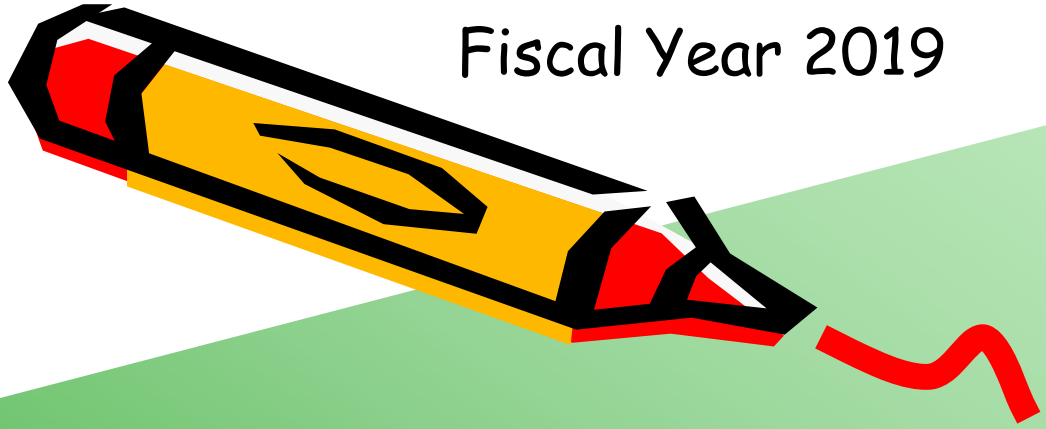


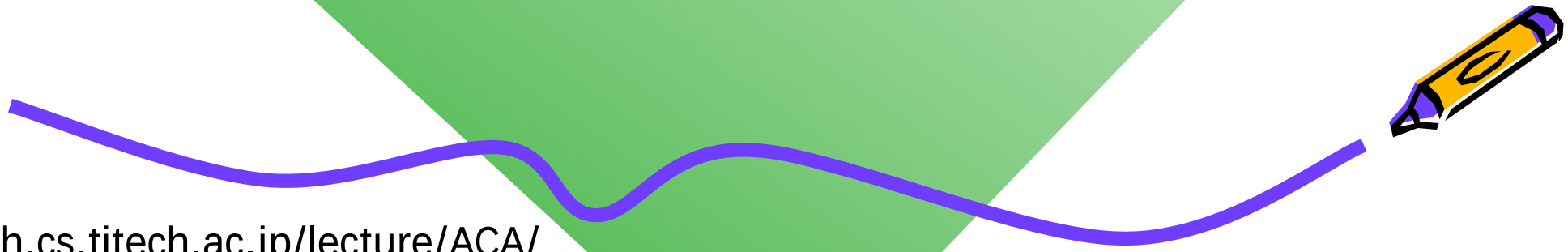
Fiscal Year 2019

Ver. 2019-12-08a

Course number: CSC.T433
School of Computing,
Graduate major in Computer Science

# Advanced Computer Architecture

## 2. Instruction Set Architecture

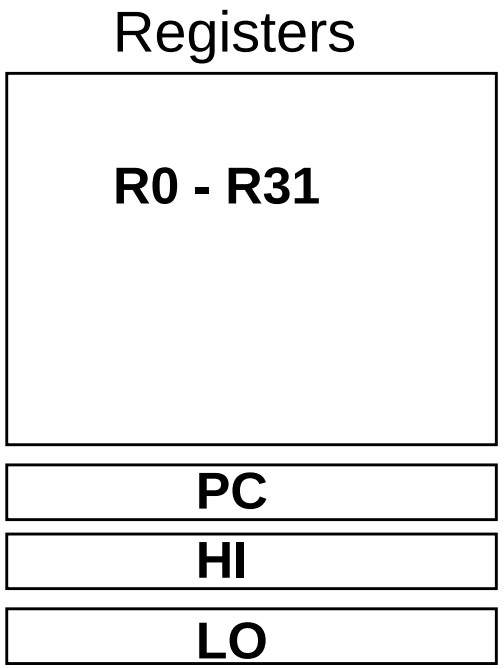


www.arch.cs.titech.ac.jp/lecture/ACA/
Room No.W936
Mon 13:20-14:50, Thr 13:20-14:50

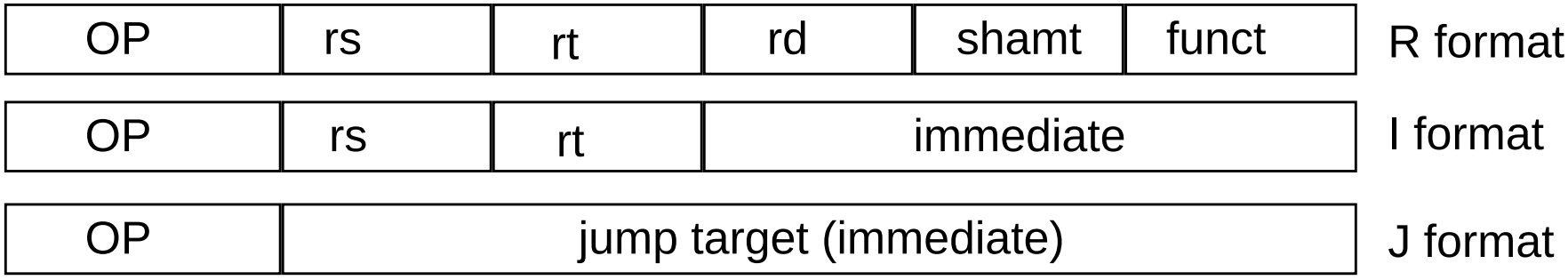
Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp

# MIPS R3000 Instruction Set Architecture (ISA)

- Instruction Categories
  - Computational
  - Load/Store
  - Jump and Branch
  - Floating Point
    - coprocessor
  - Memory Management
  - Special




**3 Instruction Formats: all 32 bits wide**



# MIPS Register Convention and ABI

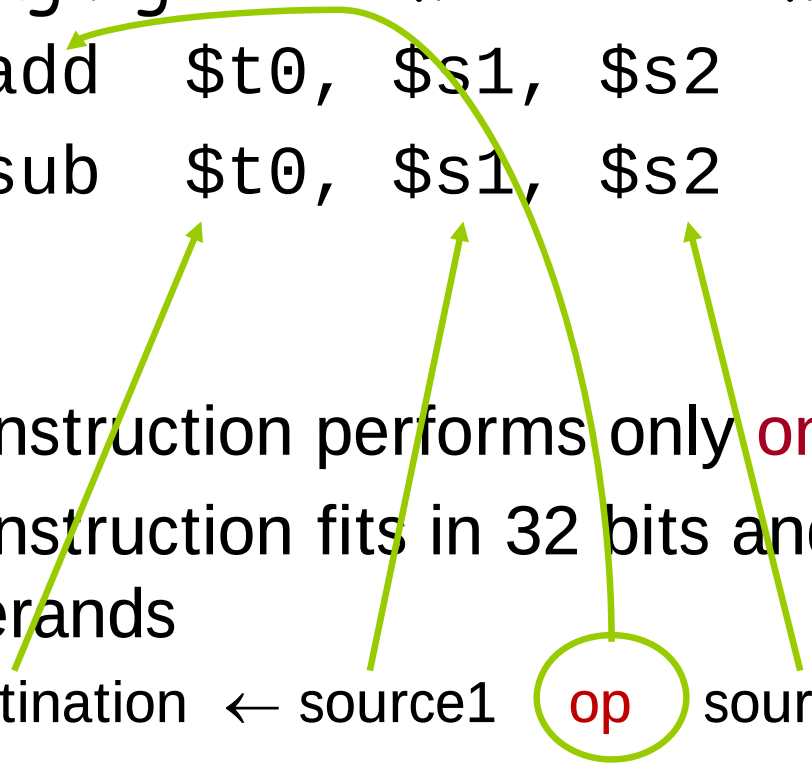
| Name | Register Number | Usage | Preserve on call? |
|---|---|---|---|
| $zero | 0 | **constant 0 (hardware)** | n.a. |
| $at | 1 | reserved for assembler | n.a. |
| $v0 - $v1 | 2-3 | returned values | no |
| $a0 - $a3 | 4-7 | **arguments** | yes |
| $t0 - $t7 | 8-15 | temporaries | no |
| $s0 - $s7 | 16-23 | saved values | yes |
| $t8 - $t9 | 24-25 | temporaries | no |
| $gp | 28 | global pointer | yes |
| $sp | 29 | stack pointer | yes |
| $fp | 30 | frame pointer | yes |
| $ra | 31 | return addr (hardware) | yes |

ABI (Application Binary Interface)

# MIPS Arithmetic Instructions

- MIPS assembly language arithmetic statement

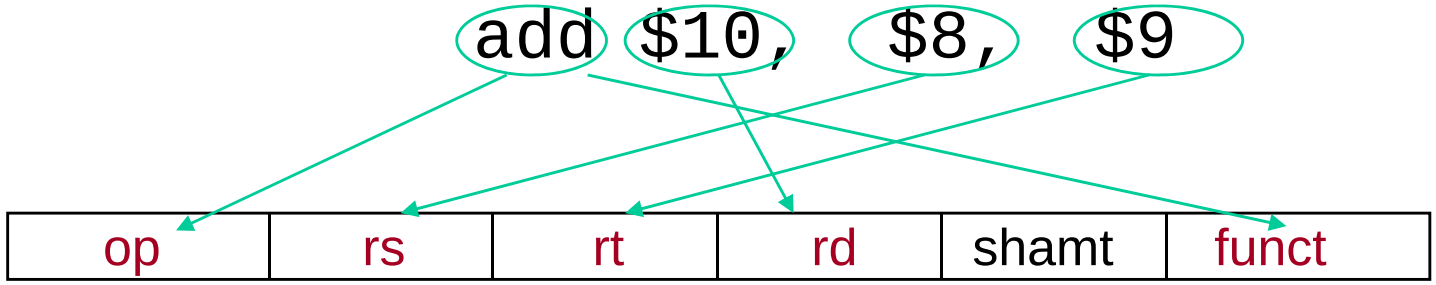
```
add  $t0, $s1, $s2
sub  $t0, $s1, $s2
```




- Each arithmetic instruction performs only one operation
- Each arithmetic instruction fits in 32 bits and specifies exactly three operands
- Operand order is fixed (destination first)
- Those operands are all contained in the datapath's register file (`$t0,$s1,$s2`) – indicated by `$`

# Machine Language - Add Instruction

- Instructions, like registers and words, are 32 bits long
- Arithmetic Instruction Format (R format):




op 6-bits opcode that specifies the operation

rs 5-bits register file address of the first source operand

rt 5-bits register file address of the second source operand

rd 5-bits register file address of the result's destination

shamt 5-bits shift amount (for shift instructions)

funct 6-bits function code augmenting the opcode

`{6'h0, 5'd8, 5'd9, 5'd10, 5'd0, 6'h20}` for `add $10, $8, $9`

# Exercise

- Compiling a C assignment Using Registers

- `f = ( g + h ) - ( i + j );`
- The variables f, g, h, i, and j are assigned to the registers $s0, $s1, $s2, $s3, and $s4, respectively. What is the compiled MIPS code?

# MIPS Immediate Instructions

- Small constants are used often in typical code
- Possible approaches?
  - put “typical constants” in memory and load them
  - create hard-wired registers (like $zero) for constants like 1
  - have special instructions that contain constants !

```
addi $sp, $sp, 4   # $sp = $sp + 4
slti $t0, $s2, 15 # $t0 = 1 if $s2<15
```

- Machine format (I format):

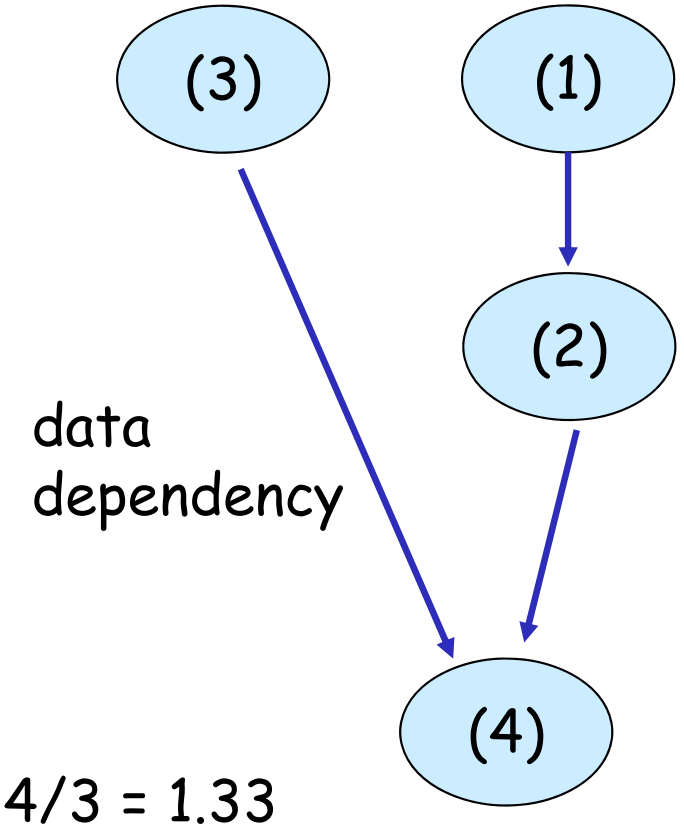

I format

- The constant is kept inside the instruction itself!
  - Immediate format limits values to the range $+2^{15}-1$ to $-2^{15}$

```
{6'h8, 5'd0, 5'd8, 16'd3} for addi $8, $0, 3
```

# Instruction Level Parallelism (ILP)

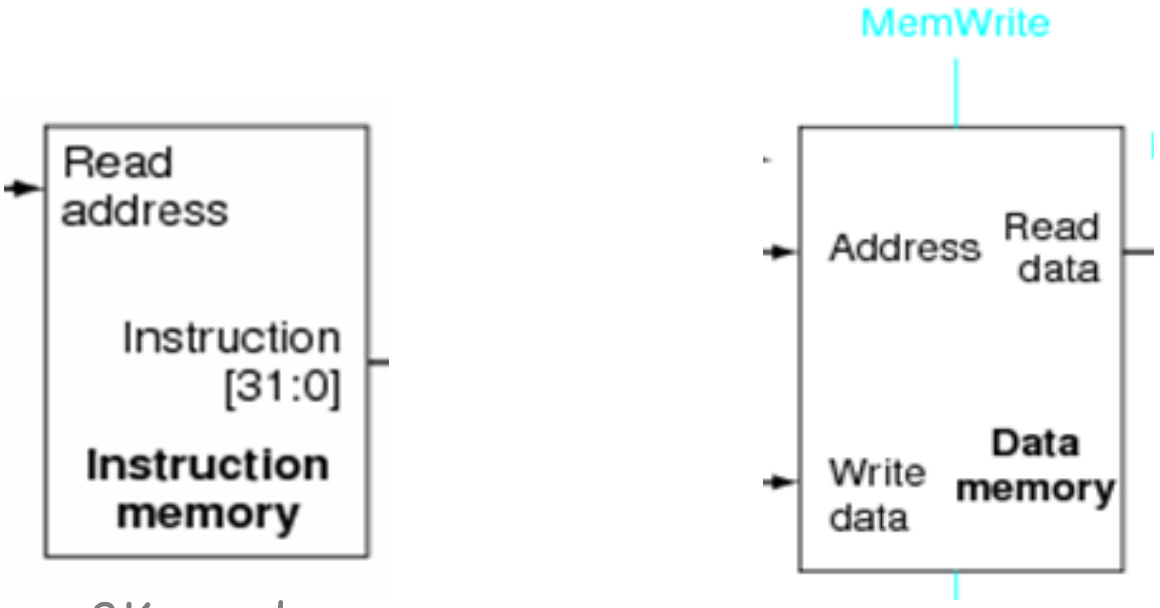
```
add   $8, $3, $5      (1)
addi  $9, $8, 1       (2)
addi  $10, $5, 1      (3)
add   $11, $10, $9    (4)
```




ILP = 4/3 = 1.33

# Computer Memory

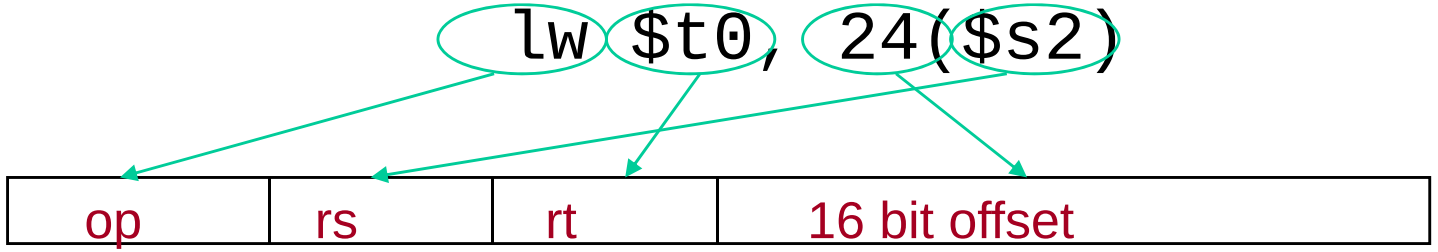
- Read-only memory (ROM)
- Random-access memory (RAM)




We use 8K word memory.

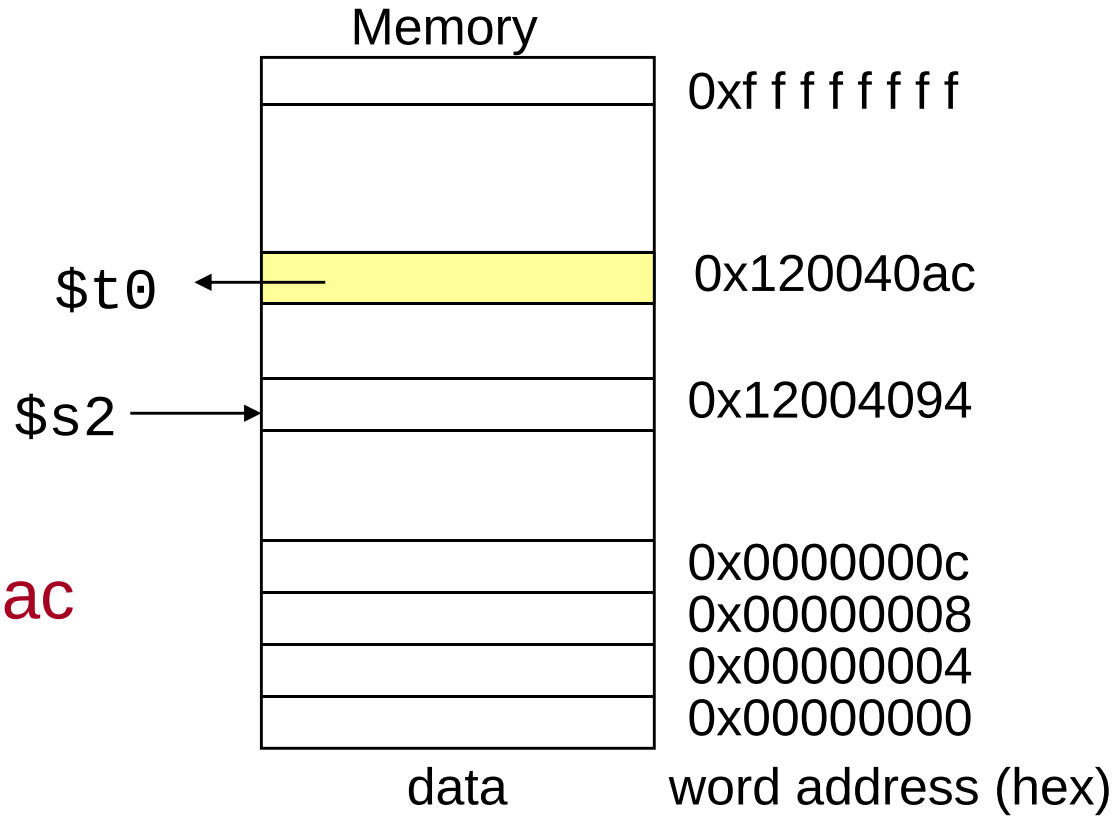
# Machine Language - Load Instruction

- Load/Store Instruction Format (I format):




$24_{10}$ + $s2 =

. . . 0001 1000
+ . . . 1001 0100
. . . 1010 1100 = 0x120040ac



# Exercise

- Compiling an Assignment When an Operand Is in Memory

- `g = h + A[8];`
- Let's assume that A is an array of 100 words and the compiler has associated the variable g and h with the registers $s1 and $s2 as before. Let's also assume that the starting address, or base address, of the array is in $s3. Compile this C assignment statement.

# MIPS Memory Access Instructions

- MIPS has two basic data transfer instructions for accessing memory

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

- The data is loaded into (lw) or stored from (sw) a register in the register file – a 5 bit address
- The memory address – a 32 bit address – is formed by adding the contents of the base address register to the offset value
  - A 16-bit field meaning 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
  - Note that the offset can be positive or negative

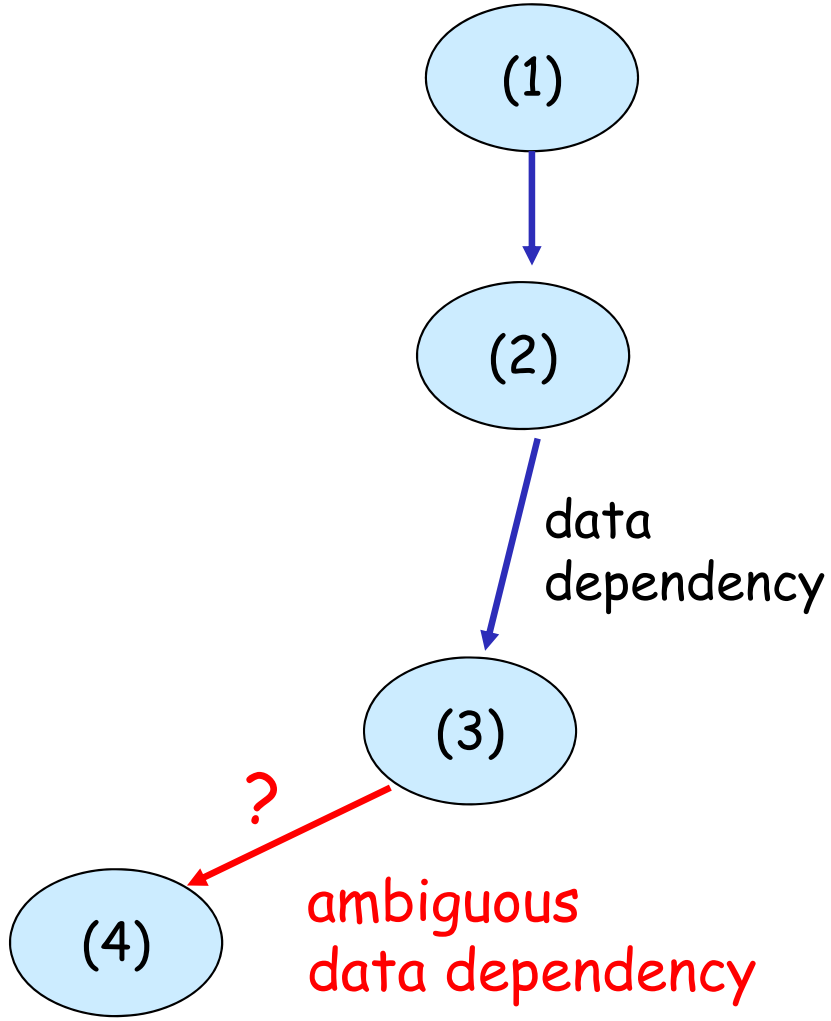
# Exercise

- Compiling Using Load and Store

- `A[12] = h + A[8];`
- Assume variable h is associated with register $s2 and base address of the array A is in $s3. What is the MIPS assembly code for the C assignment statement?

# Instruction Level Parallelism (ILP)

```
lw    $t0, 32($s3)      (1)
add   $t0, $s2, $t0     (2)
sw    $t0, 48($s3)      (3)
        ?
lw    $t1, 32($s4)      (4)
```



# MIPS Control Flow Instructions

- MIPS conditional branch instructions:

```
bne $s0, $s1, Lbl   # go to Lbl if $s0≠$s1
beq $s0, $s1, Lbl   # go to Lbl if $s0=$s1
```

  - Ex: `if (i==j) h = i + j;`

```
    bne $s0, $s1, Lbl1
    add $s3, $s0, $s1
Lbl1:   ...
```

  - Instruction Format (I format):

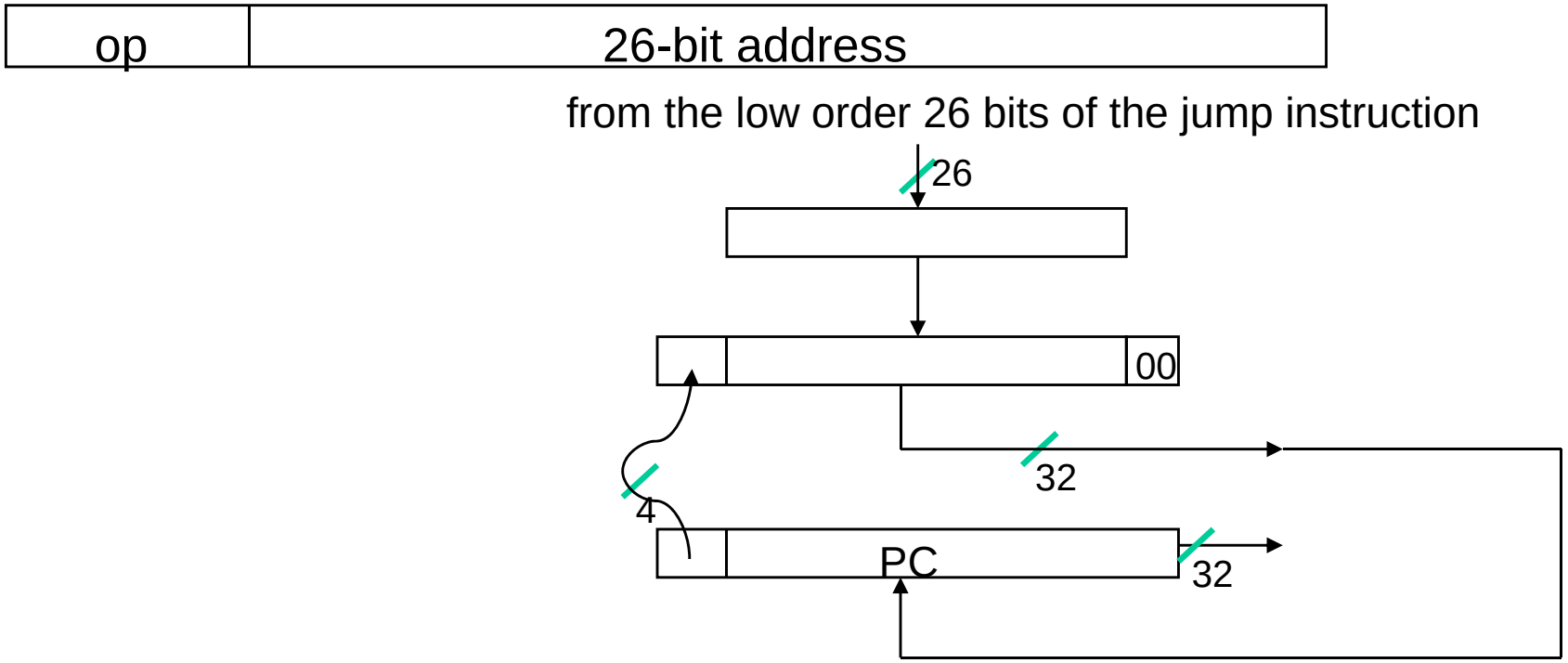

  - How is the branch destination address specified?

# Other Control Flow Instructions

- MIPS also has an **unconditional branch** instruction or jump instruction:

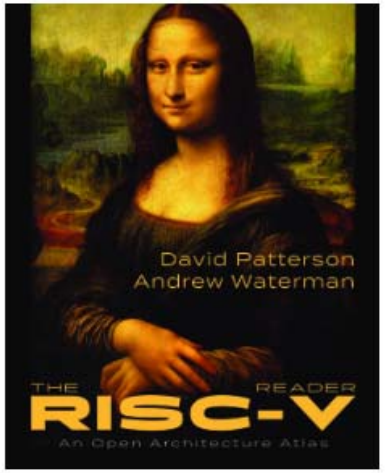
```
j  label     # go to label
```

- Instruction Format (J Format):



# RISC – Reduced Instruction Set Computer

- RISC philosophy
  - fixed instruction lengths
  - load-store instruction sets
  - limited addressing modes
  - limited operations
- RISC-I, MIPS, DEC Alpha, ARM, RISC-V, …



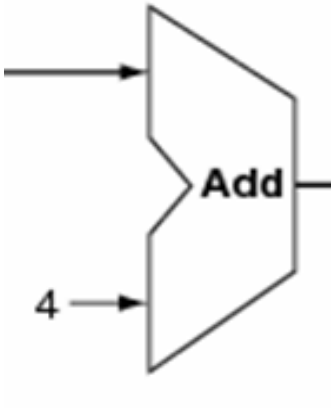
# CISC - Complex Instruction Set Computer

- CISC philosophy
  - ! fixed instruction lengths
  - ! load-store instruction sets
  - ! limited addressing modes
  - ! limited operations
- DEC VAX11, Intel 80x86, ...

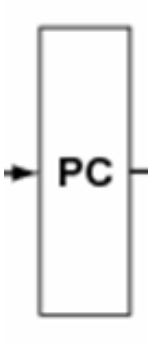
# Single-cycle implementation of processors

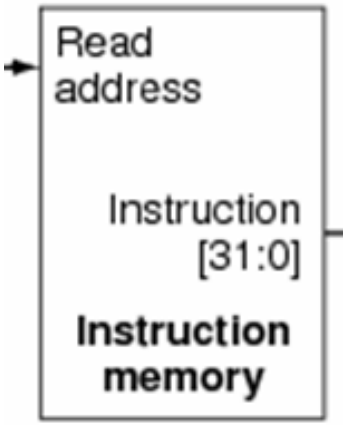
- Single-cycle implementation also called single clock cycle implementation is the implementation in which an instruction is executed in one clock cycle. While easy to understand, it is too slow to be practical.

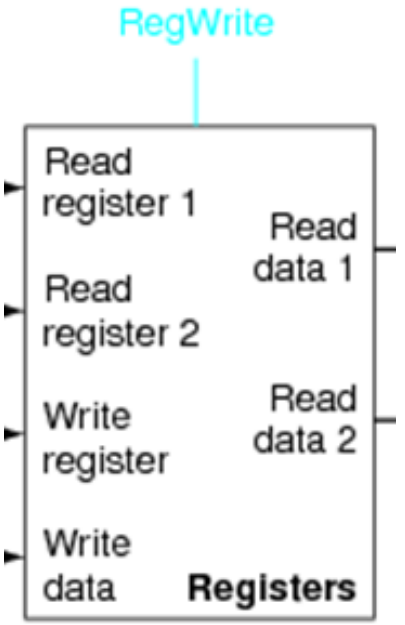
# Some building blocks of processor datapath







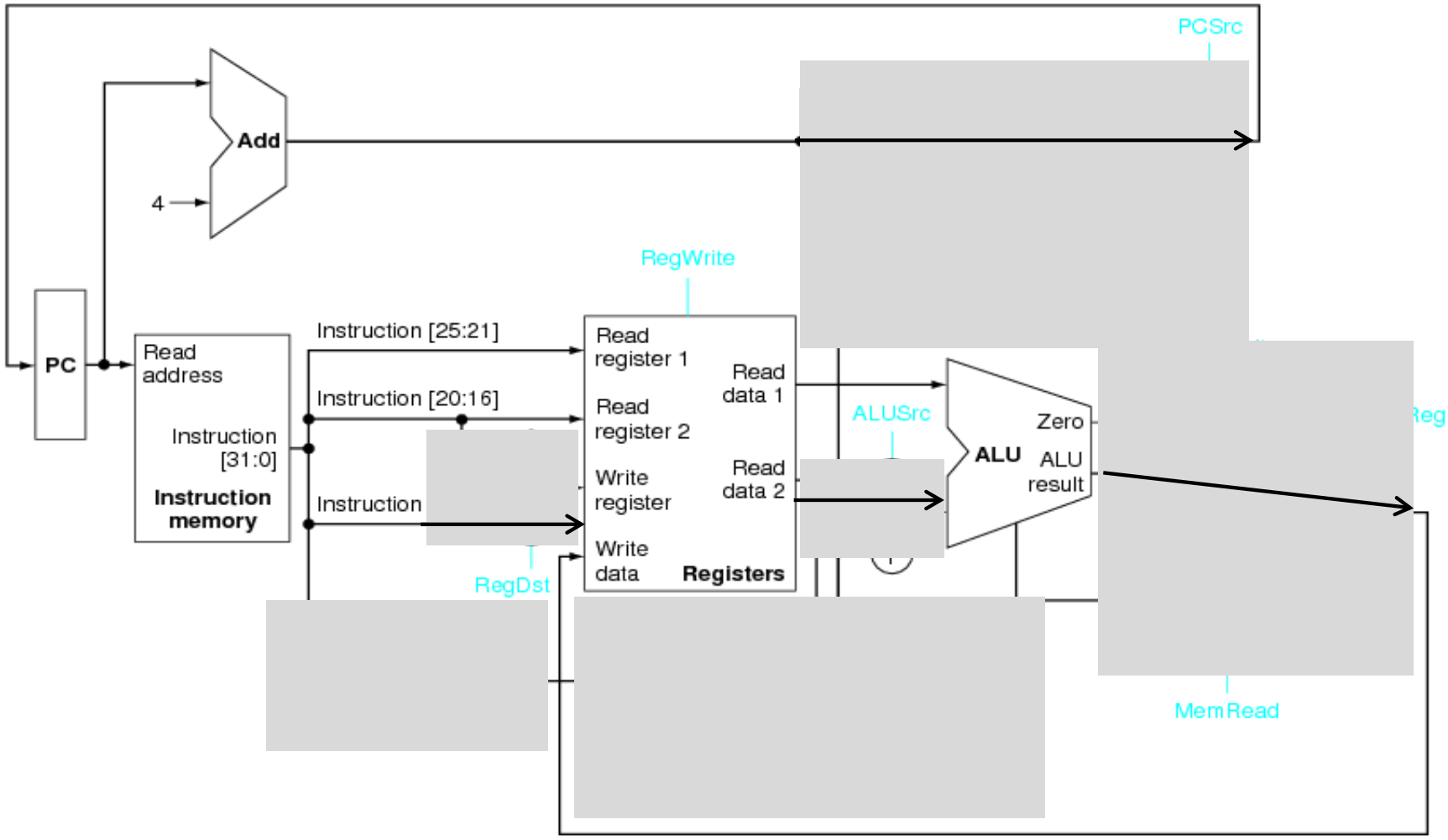




We use 8K word memory.

# Datapath of single-cycle processor supporting ADD

| | IR[25:21] | IR[20:16] | IR[15:11] | | |
|---|---|---|---|---|---|
| op | rs | rt | rd | shamt | funct |

```
0x800    add $t0, $s1, $s2   [ add $8, $17, $18 ]
```




**$17 = 3**
**$18 = 4**

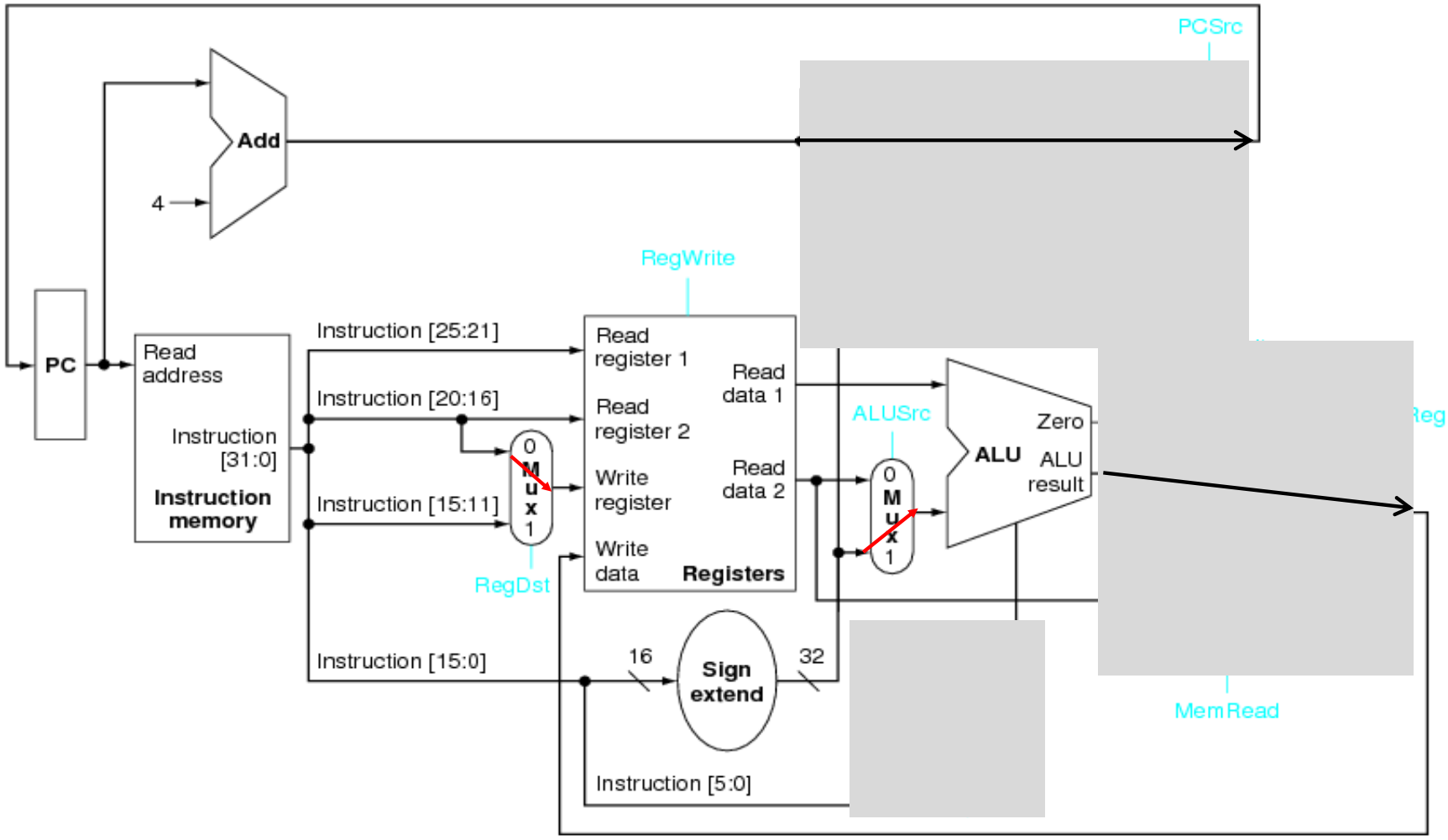
# Datapath of processor supporting ADD and ADDI

IR[25:21] IR[20:16]

| op | rs | rt | 16 bit immediate |
|---|---|---|---|

I format

```
0x804   addi $t1, $t0, 3      [ addi $9, $8, 3 ]
```




**$8 = 7**

# Assignment 2

1. Design a single-cycle processor supporting MIPS add, addi instructions in Verilog HDL. Please download proc01.v from the support page and refer it.
2. Verify the behavior of designed processor using following assembly code
   - `add  $0,  $0,  $0   # NOP {6'h0, 5'd0, 5'd0, 5'd0, 5'd0, 6'h20}`
   - `addi $t0, $zero, 3  # {6'h8, 5'd0, 5'd8, 16'd3}`
   - `addi $t1, $zero, 5  # {6'h8, 5'd0, 5'd9, 16'd5}`
   - `add  $t2, $t0, $t1  # {6'h0, 5'd8, 5'd9, 5'd10, 5'd0, 6'h20}`
3. Submit a report printed on A4 paper at the beginning of the next lecture.
   - The report should include a block diagram, a source code in Verilog HDL, and obtained waveforms of your design.

# Waveform of proc01