

# EE309(S2): Microprocessors

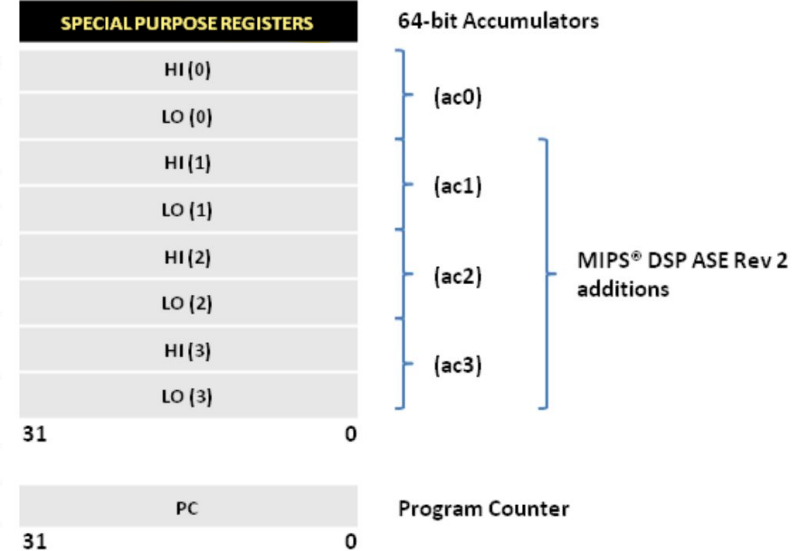
Spring 2025

[Week#9 Slides]

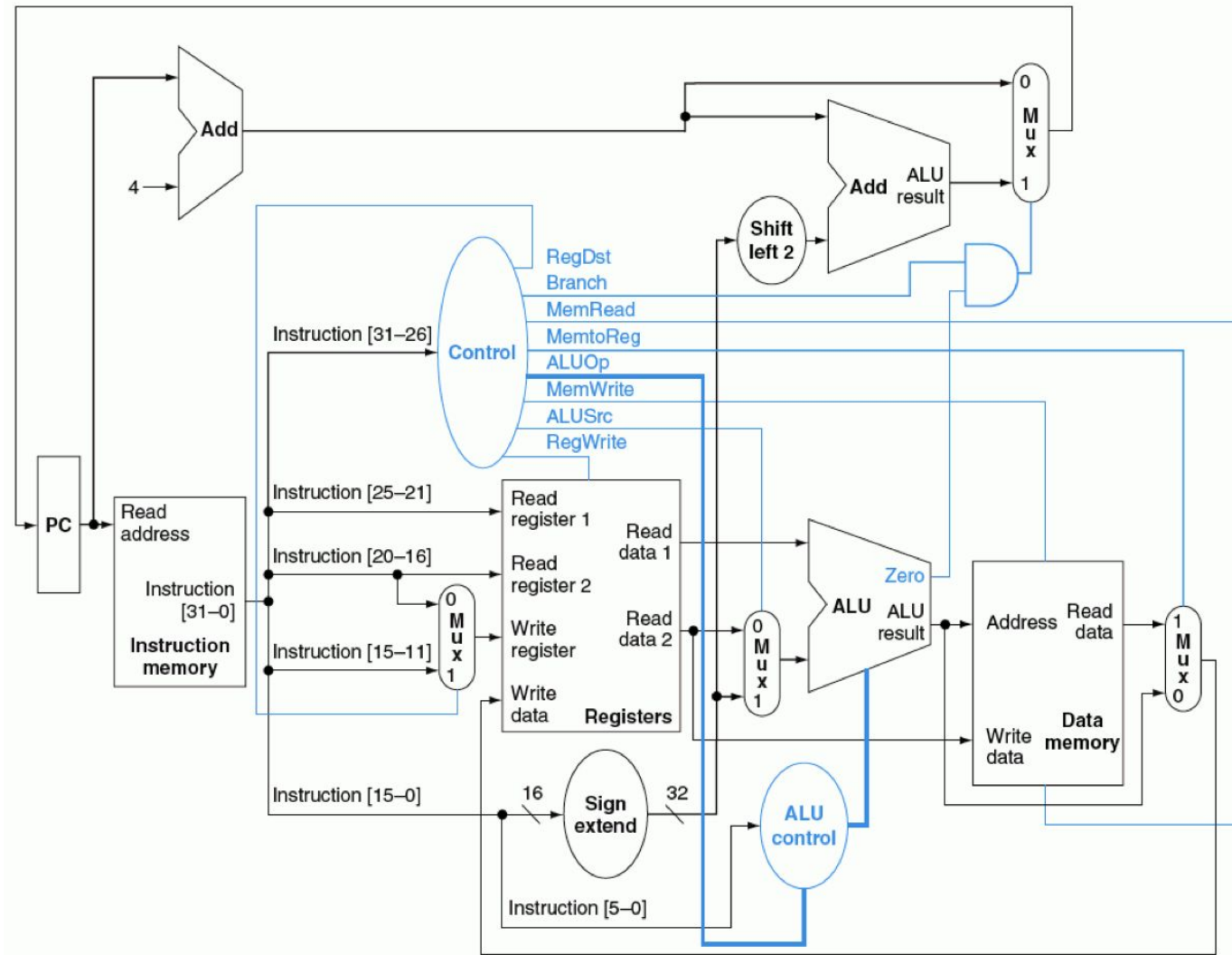
Instructor: Shalabh Gupta

# Register File

Register Number	Software Name	Use
\$0	zero	Always 0 when read.
\$1	at	Assembler temporary variable. Do not use the \$at register from source code unless you really know what you are doing.
\$2-\$3	v0-v1	Return value from functions.
\$4-\$7	a0-a3	Used for passing arguments to functions.
\$8-\$15	t0-t7	Temporary registers used by compiler for expression evaluation. Values not saved across function calls.
\$16-\$23	s0-s7	Temporary registers whose values are saved across function calls.
\$24-\$25	t8-t9	Temporary registers used by compiler for expression evaluation. Values not saved across function calls.
\$26-\$27	k0-k1	Reserved for interrupt/trap handler.
\$28	gp	Global Pointer.
\$29	sp	Stack Pointer.
\$30	fp or s8	Frame Pointer if needed. Additional temporary saved register if not.
\$31	ra	Return address for functions.



# MIPS Controller and Datapath (without pipelining)



# Seminars / Hardware project to make up for low marks in up to 4 or 5 quizzes

## Eligibility:

- Students who have been attending classes
- Students who have been giving quizzes

## The following students are NOT eligible:

- Students who cheated by submitting quizzes from outside the exam hall
- Among these students, those who have not yet confessed, will also get grade penalties or will be reported to the academic office
- Students with poor attendance record (<80% attendance)

# Optional Seminar and Hardware Project

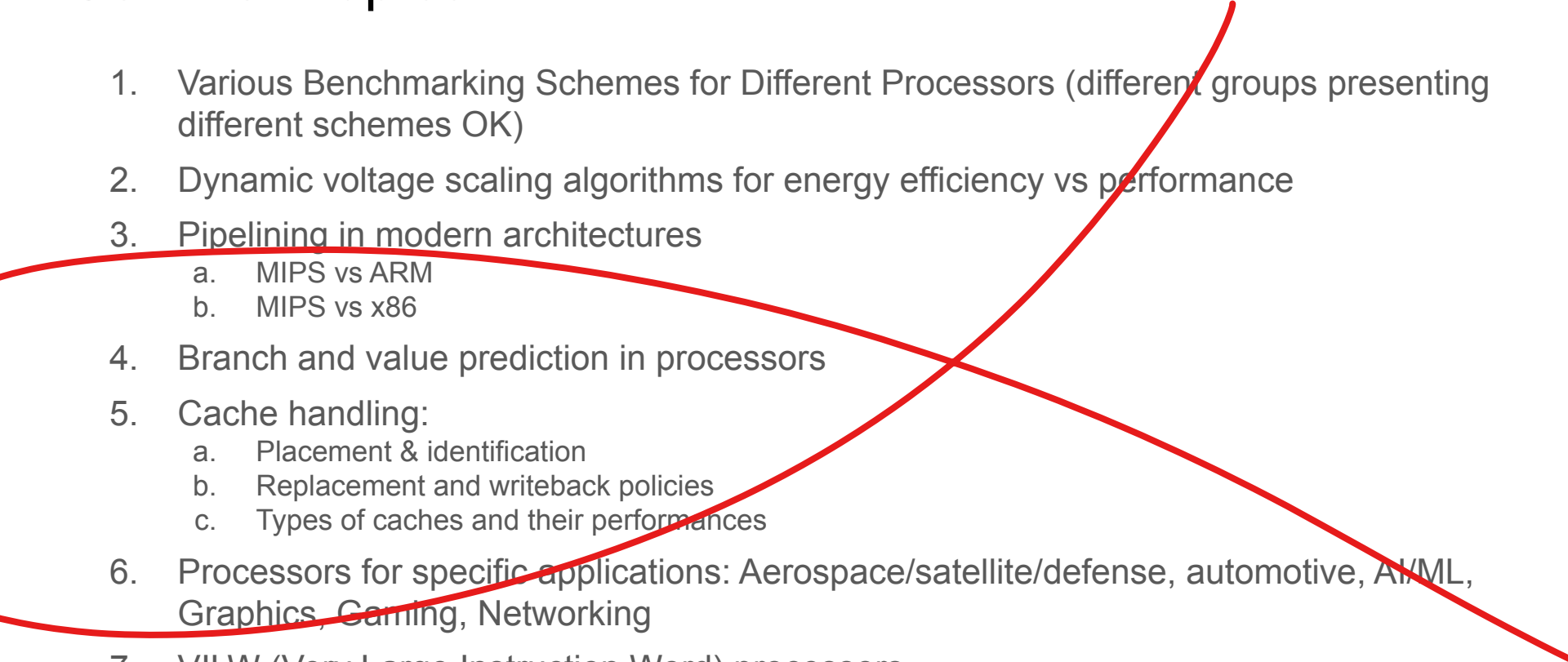
Seminars (worth 4 quizzes):

- 20 min presentation by each group (strict time limit).
- Groups of three (each member has to speak and answer)
- Marks depend on how much new stuff the students can learn from your presentation, and your preparation & presentation level.

Hardware project (worth 5 quizzes):

- Demos can be done later (after exams)
- Groups of 3
- Harmonic / intermodulation (non-linearity) measurement

# Seminar Topics

1. Various Benchmarking Schemes for Different Processors (different groups presenting different schemes OK)
  2. Dynamic voltage scaling algorithms for energy efficiency vs performance
  3. Pipelining in modern architectures
    - a. MIPS vs ARM
    - b. MIPS vs x86
  4. Branch and value prediction in processors
  5. Cache handling:
    - a. Placement & identification
    - b. Replacement and writeback policies
    - c. Types of caches and their performances
  6. Processors for specific applications: Aerospace/satellite/defense, automotive, AI/ML, Graphics, Gaming, Networking
  7. VILW (Very Large Instruction Word) processors
  8. Other topics proposed by you (open ended)
- 

# MIPS Instruction Format

(Bits) R – Type	(31-26) Opcode	(25-21) rs	(20-16) rt	(15-11) rd	(10-6) shift amount	(5-0) function
(Bits) – Type	(31-26) Opcode	(25-21) rs	(20-16) rt	(15-0) Immediate		
(Bits) J –Type	(31-26) Opcode	(25-0) Psuedodirect jump address				

Examples:

R-Type: <opcode> rd, rs, rt

add \$t0, \$t1, \$t2 # \$t0 = \$t1 + \$t2

I-Type (arithmetic/logic): <opcode> rt, rs, immediate

addi \$t0, \$t1, 100 # \$t0 = \$t1 + 100

I-Type (memory): <opcode> rt, offset(rs)

sw \$t0, 20(\$t1) # Store the word in target register \$t0 to the address \$t1 + 20

lw \$t0, 20(\$t1) # Load the word in target register \$t0 from the address \$t1 + 20

# MIPS Instructions: Logical

Instruction	Example	Meaning	Comments
<b>and</b>	<code>and \$1, \$2, \$3</code>	$\$1 = \$2 \& \$3$	Bitwise AND
<b>or</b>	<code>or \$1, \$2, \$3</code>	$\$1 = \$2   \$3$	Bitwise OR
<b>and immediate</b>	<code>andi \$1, \$2, 100</code>	$\$1 = \$2 \& 100$	Bitwise AND with immediate value
<b>or immediate</b>	<code>or \$1, \$2, 100</code>	$\$1 = \$2   100$	Bitwise OR with immediate value
<b>shift left logical</b>	<code>sll \$1, \$2, 10</code>	$\$1 = \$2 \ll 10$	Shift left by constant number of bits
<b>shift right logical</b>	<code>srl \$1, \$2, 10</code>	$\$1 = \$2 \gg 10$	Shift right by constant number of bits



# MIPS

## Instruction:

## Data

## Transfer

Instruction	Example	Meaning	Comments
load word	lw \$1,100(\$2)	\$1=Memory[\$2+100]	Copy from memory to register
store word	sw \$1,100(\$2)	Memory[\$2+100]=\$1	Copy from register to memory
load upper immediate	lui \$1,100	\$1=100x2 <sup>16</sup>	Load constant into upper 16 bits. Lower 16 bits are set to zero.
load address	la \$1,label	\$1=Address of label	<i>Pseudo-instruction</i> (provided by assembler, not processor!) Loads computed address of label (not its contents) into register
load immediate	li \$1,100	\$1=100	<i>Pseudo-instruction</i> (provided by assembler, not processor!) Loads immediate value into register
move from hi	mghi \$2	\$2=hi	Copy from special register hi to general register
move from lo	mflo \$2	\$2=lo	Copy from special register lo to general register
move	move \$1,\$2	\$1=\$2	<i>Pseudo-instruction</i> (provided by assembler, not processor!) Copy from register to register.

# MIPS

## Instructions: Conditional Branch

Instruction	Example	Meaning	Comments
branch on equal	beq \$1, \$2, 100	if(\$1==\$2) go to PC+4+100	Test if registers are equal
branch on not equal	bne \$1, \$2, 100	if(\$1!=\$2) go to PC+4+100	Test if registers are not equal
branch on greater than	bgt \$1, \$2, 100	if(\$1>\$2) go to PC+4+100	<i>Pseduo-instruction</i>
branch on greater than or equal	bge \$1, \$2, 100	if(\$1>=\$2) go to PC+4+100	<i>Pseduo-instruction</i>
branch on less than	blt \$1, \$2, 100	if(\$1<\$2) go to PC+4+100	<i>Pseduo-instruction</i>
branch on less than or equal	ble \$1, \$2, 100	if(\$1<=\$2) go to PC+4+100	<i>Pseduo-instruction</i>

# MIPS Instructions: Comparison

Instruction	Example	Meaning	Comments
<b>set on less than</b>	<code>slt \$1,\$2,\$3</code>	if( $\$2 < \$3$ ) $\$1=1$ ; else $\$1=0$	Test if less than. If true, set $\$1$ to 1. Otherwise, set $\$1$ to 0.
<b>set on less than immediate</b>	<code>slti \$1,\$2,100</code>	if( $\$2 < 100$ ) $\$1=1$ ; else $\$1=0$	Test if less than. If true, set $\$1$ to 1. Otherwise, set $\$1$ to 0.

# MIPS Instructions

## Unconditional Jump

Instruction	Example	Meaning	Comments
jump	j 1000	go to address 1000	Jump to target address
jump register	jr \$1	go to address stored in \$1	For switch, procedure return
jump and link	jal 1000	\$ra=PC+4; go to address 1000	Use when making procedure call. This saves the return address in \$ra

# Pseudo Instructions & Examples

Instructions that the MIPS assembler recognizes (although they are not MIPS machine instructions) -- but can be implemented using one or more MIPS machine instructions

Task	Pseudo-Instruction	Programmer Writes	Assembler Translates To
Move the contents of one register to another.	<code>move &lt;dest&gt; &lt;source&gt;</code>	<code>move \$t0, \$s0</code>	<code>addu \$t0, \$zero, \$s0</code>
Load a constant into a register. (Negative values are handled slightly differently.) "li" stands for load immediate.	<code>li &lt;dest&gt; &lt;immed&gt;</code>	<code>li \$s0, 10</code>	<code>ori \$s0, \$zero, 10</code>
Load the <i>word</i> stored in a named memory location into a register. <b>Variable</b> is a label that the programmer has attached to a memory location. 12 is the offset of that memory location from the beginning of the data segment. It is calculated by the assembler for you.	<code>lw &lt;reg&gt; &lt;label&gt;</code>	<code>lw \$s0, variable</code>	<code>lui \$at, 0x1001</code> <code>lw \$s0, 12(\$at)</code>

# Pseudo-Instructions: Examples

Task	Pseudo-Instruction	Programmer Writes	Assembler Translates To
Load the <i>address</i> of a named memory location into a register. <i>Value</i> is a label that the programmer has attached to a memory location. 16 is the offset of that memory location from the beginning of the data segment. It is calculated by the assembler for you.	<code>la &lt;dest&gt; &lt;label&gt;</code>	<code>la \$s0, variable</code>	<code>lui \$at, 0x1001</code> <code>ori \$s0, \$at, 16</code>
If $r1 < r2$ , branch to label.	<code>blt &lt;r1&gt;, &lt;r2&gt;, &lt;label&gt;</code>	<code>blt \$t0, \$t1, for_exit</code>	<code>slt \$at, \$t0, \$t1</code> <code>bne \$at, \$zero, for_exit</code>
If $r1 \leq r2$ , branch to label.	<code>ble &lt;r1&gt;, &lt;r2&gt;, &lt;label&gt;</code>	<code>ble \$t0, \$t1, for_exit</code>	<code>slt \$at, \$t1, \$t0</code> <code>beq \$at, \$zero, for_exit</code>
If $r1 > r2$ , branch to label.	<code>bgt &lt;r1&gt;, &lt;r2&gt;, &lt;label&gt;</code>	<code>bgt \$t0, \$t1, for_exit</code>	<code>slt \$at, \$t1, \$t0</code> <code>bne \$at, \$zero, for_exit</code>
If $r1 \geq r2$ , branch to label.	<code>bge &lt;r1&gt;, &lt;r2&gt;, &lt;label&gt;</code>	<code>bge \$t0, \$t1, for_exit</code>	<code>slt \$at, \$t0, \$t1</code> <code>beq \$at, \$zero, for_exit</code>

Source: <https://matthews.sites.truman.edu/files/2019/11/pseudoinstructions.pdf>

**Practice Assignment: Look up for other the pseudo-instructions (on the internet) for the basic MIPS ISA and try implementing all using real MIPS machine instructions**

# Pipelining Hazards

Situations that can cause delays or inefficiencies in the execution of instructions within a pipelined processor. Three main types of hazards are there:

- a. **Structural Hazards** Same hardware is needed by more than one instructions simultaneously
- b. **Data Hazards** The result needed for one instruction is not yet available from the previous instruction
- c. **Control Hazards (Branching Hazards)** Branching affects the flow of instructions