CS2100 - L10 - Processors: Datapath

Week 5+6

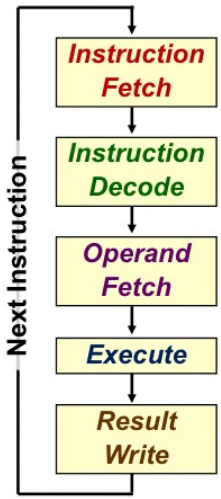
10.1 - Generic Execution Stages

10.2 - MIPS Execution Stages (FDEMW)

- [F] Fetch
- [D] Decode & Operation Fetch
- [E] Execute: ALU
- [M] Memory Access
- [W] Result Write

10.3 - Constructing Datapath

10.1 - Generic Execution Stages



- **Fetch:**
  - Get instruction from memory
  - Address is in **P**rogram **C**ounter (PC) Register
- **Decode:**
  - Find out the operation required
- **Operand Fetch:**
  - Get operand(s) needed for operation
- **Execute:**
  - Perform the required operation
- **Result Write (Store):**
  - Store the result of the operation

10.2 - MIPS Execution Stages (FDEMW)

5-STAGE MIPS EXECUTION

① same duration
② same kind of operation ⇒ mechanism

- Design changes:
  - Merge *Decode* and *Operand Fetch* – Decode is simple for MIPS
  - Split *Execute* into ***ALU*** (Calculation) and ***Memory Access***

	`add $3, $1, $2`	`lw $3, 20( $1 )`	`beq $1, $2, label`
F **Fetch**	Read inst. at [PC]	Read inst. at [PC]	Read inst. at [PC]
D **Decode & Operand Fetch**	○Read [**$1**] as *opr1* ○Read [**$2**] as *opr2*	○Read [**$1**] as *opr1* ○Use ***20*** as *opr2*	○Read [**$1**] as *opr1* ○Read [**$2**] as *opr2*
E **ALU**	*Result* = *opr1* + *opr2*	*MemAddr* = opr1 + opr2	*Taken* = (*opr1* == *opr2*)? *Target* = **PC** + Label*
M **Memory Access**		Use *MemAddr* to read from memory	
W **Result Write**	*Result* stored in **$3**	*Memory* data stored in **$3**	if (*Taken*) **PC** = *Target*

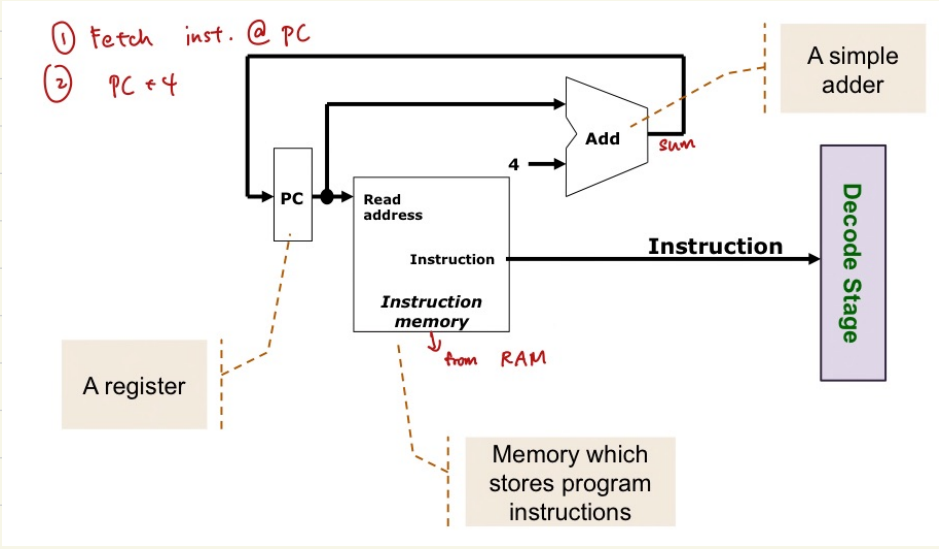
- Simplest possible implementation of a subset of the core MIPS ISA:
  - **Arithmetic and Logical operations**
    - `add`, `sub`, `and`, `or`, `addi`, `andi`, `ori`, `slt`
  - **Data transfer instructions**
    - `lw`, `sw`
  - **Branches**
    - `beq`, `bne`
- Shift instructions (`sll`, `srl`) and J-type instructions (`j`) will not be discussed:
  - Left as exercises ☺

[F] Fetch

①

Fetch Stage: Requirements

- Instruction **Fetch Stage**:
  1. Use the **Program Counter** (**PC**) to fetch the instruction from **memory**
     - PC is implemented as a special register in the processor
  2. **Increment** the PC by 4 to get the address of the next instruction:
     - How do we know the next instruction is at PC+4?
     - Note the exception when branch/jump instruction is executed
- Output to the next stage (**Decode**):
  - The instruction to be executed

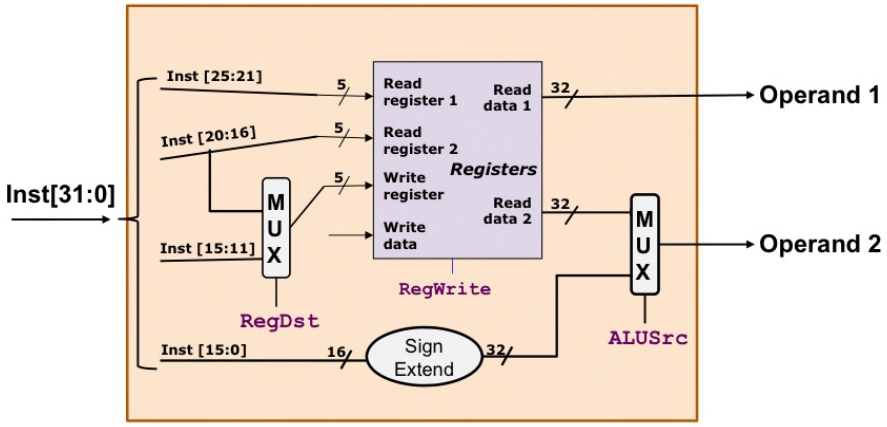


[D] Decode & Operation Fetch

(2)

Decode Stage: Requirement

- Instruction **Decode Stage**:
  - Gather data from the instruction fields:
    1. Read the **opcode** to determine instruction type and field lengths
    2. Read data from all necessary registers
       - Can be two (e.g. `add`), one (e.g. `addi`) or zero (e.g. `j`)
- Input from previous stage (**Fetch**):
  - Instruction to be executed
- Output to the next stage (**Execute**):
  - Operation and the necessary operands



[E] Execute : ALU

③ ALU Stage: Requirement

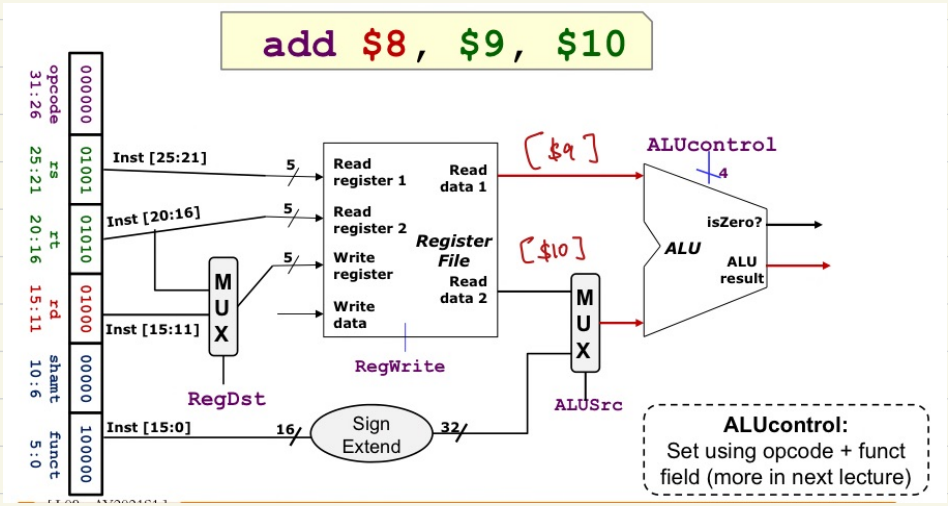
- Instruction **ALU Stage**:
  - **ALU** = **A**rithmetic-**L**ogic **U**nit
  - Perform the real work for most instructions here
    - Arithmetic (e.g. `add`, `sub`), Shifting (e.g. `sll`), Logical (e.g. `and`, `or`)
    - Memory operation (e.g. `lw`, `sw`): Address calculation
    - Branch operation (e.g. `bne`, `beq`): Perform register comparison and target address calculation
- Input from previous stage (**Decode**):
  - Operation and Operand(s)
- Output to the next stage (**Memory**):
  - Calculation result

Categorise into :

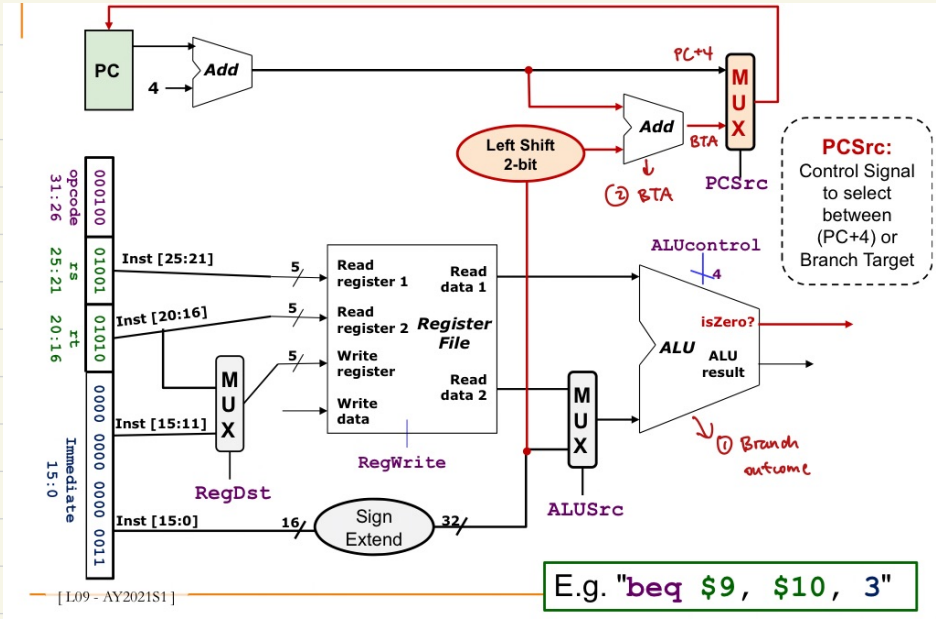
① Non-branch instructions

② Branch instructions

[E] Non-Branch Instructions



[E] Branch Instructions



↳ [$9] == [$10] ⇒ [$9] - [$10] isZero?

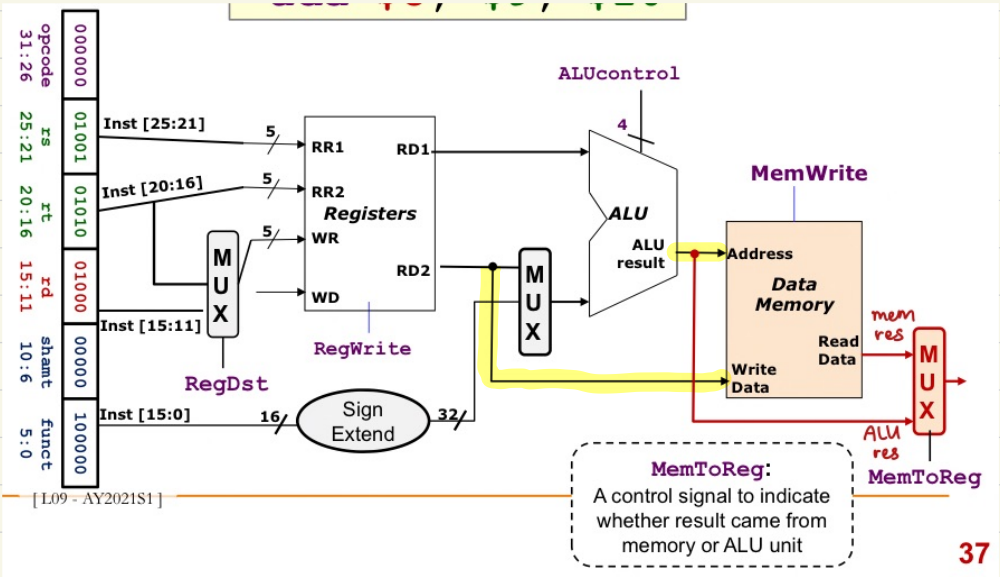
[M] Memory Access

(4)

Memory Stage: Requirement

Only for lw/sw/lb/sb !!!

- Instruction **Memory Access Stage**:
  - Only the load and store instructions need to perform operation in this stage:
    - Use memory address calculated by ALU Stage
    - Read from or write to data memory
  - All other instructions remain idle
    - Result from ALU Stage will pass through to be used in Result Store stage if applicable
- Input from previous stage (**ALU**):
  - Computation result to be used as memory address (if applicable)
- Output to the next stage (**Result Write**):
  - Result to be stored (if applicable)

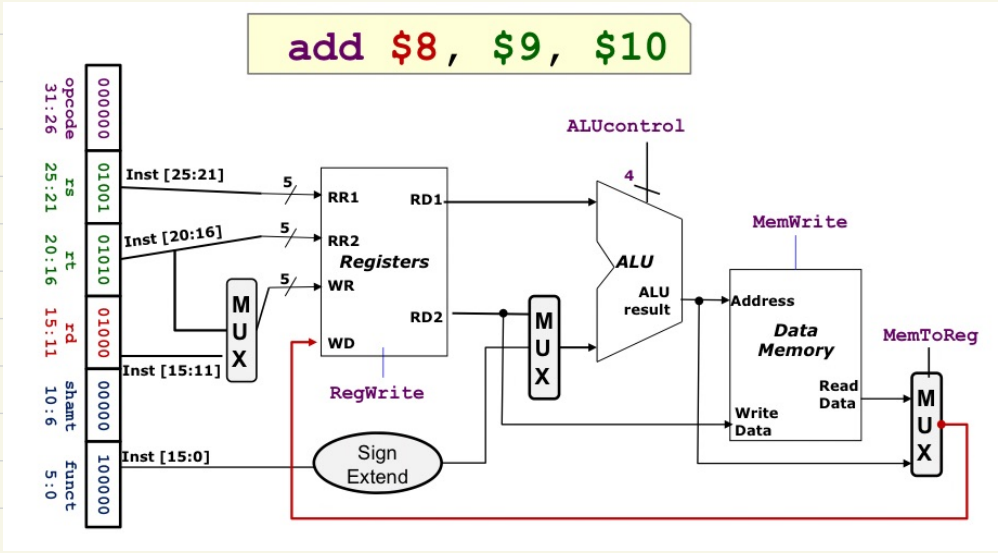


[W] Result Write

5

Result Write Stage: Requirement

- Instruction **Register Write Stage**:
  - Most instructions write the result of some computation into a register
    - Examples: arithmetic, logical, shifts, loads, set-less-than
    - Need destination register number and computation result
  - Exceptions are stores, branches, jumps:
    - There are no result to be written
    - → These instructions remain idle in this stage
- Input from previous stage (**Memory**):
  - Computation result either from memory or ALU



10.3 – Constructing Datapath

Shifting and jumping are not supported!