Simple Operation of MIPS

Consider the operation of MIPS without pipelining. Every MIPS instruction can be executed in at most 5 clock cycles - representing five phases of operation:

1. Instruction Fetch (IF):
2. Instruction decode/register fetch cycle (ID):
3. Execution/effective address cycle (EX):
4. Memory access/branch completion cycle (MEM):
5. Write-back cycle (WB):

---

1. **MIPS Instruction Fetch (IF):**

```
IR  <- Mem[PC]          ; Instruction Register, Program Counter
NPC <- PC + 4           ; Next Program Counter
```

2. **MIPS Instruction decode/register fetch cycle (ID):**

```
A   <- Regs[IR(6..10)]
B   <- Regs[IR(11..15)]
Imm <- sign extended IR(16..31)
```

decoding -> will determine the following sequence based on instruction

---

3. **MIPS Execution/effective address cycle (EX):**

one of the following depending on the instruction decoded:

a. memory reference:

```
ALUoutput <- A + Imm
```

b. register-register ALU instruction:

```
ALUoutput <- A op B
```

c. register-immediate ALU operation:

```
ALUoutput <- A op Imm
```

d. branch:

```
ALUoutput <- NPC + Imm
Cond <- A op 0
```

---

4. **MIPS Memory access/branch completion cycle (MEM):**

one of the following based on the instruction decoded:

a. Memory load

```
LMD <- Mem[ALUoutput]
```

b. Memory store

```
Mem[ALUoutput] <- B
```

c. Branch

```
if(cond) PC <- ALUoutput
  else   PC <- NPC
```

---

5. **MIPS Write-back cycle (WB):**

one of the following depending on the instruction decoded:

a. register-register ALU instruction

```
Regs[IR(16..20)] <- ALUoutput
```

b. register-immediate ALU instruction

```
Regs[IR(11..15)] <- ALUoutput
```

c. load instruction

```
Regs[IR(11..15)] <- LMD
```

d. branch instruction - does not have a WB cycle