1. Arthmetic must stall on reading from r/m which is in Setup Operands.
   * This limits us in when we can read in which in done is setup operands.
2. Mem Stores should also occur as late as possible and occur during Setup Operands. These don’t stall like dependencies stalls back we only need to wait whenever the instruction in the writeback stage needs memory, not any instruction that is writing to a source.
3. Both Arthmetic/Mem Stalls rely on not being able to get the right source. A small fix would be to do data forwarding. The large fix is to do out of order execution.
4. JMP will change the EIP so once we decode Jmp we can no longer fetch until we update EIP.
   * JMP may need to access memory, so at earliest, we can only update EIP in Setup Operands.
   * Fix is target prediction cache which doesn’t seem to be useful enough to justify cost.
5. For JMPcc, not only can we no longer fetch until it updates EIP, but it must wait for the previous instruction to get to WB and set CC. In this case, we stall the JMPcc if any valid instruction in the pipeline sets cc, and then once the JMPcc is no lingered stalled, then the fetch of the next instruction must be stalled.
   * CC forwarding will only save us 1 stage in the pipeline. Which is about a 1/7 speedup (not world changing)
   * The large fix is to use a branch predictor and flush if in correct.
6. CMov is unlike JMPcc because it doesn’t change EIP so we can keep fetching. At the latest it needs the CC at writeback. At this time any previous instruction will have updated CC and we can read from it directly.
7. The length of a pipeline doesn’t matter, only the clock speed at which each stage can finish. The length of stall depends on how many stages we need to wait.
8. Movement in the pipeline only happens when the latches are valid, “trash operations are not performed” due to control signals.
9. All stalls must be checked with valid before sending to other stages.
10. Assume Register file has 2 write port and 2 read ports the 2 read ports will be muxed to allow either generate address or setup operand
11. Setting flags later is better because I can do more work before I have to stall. Flags can be retrieved in any stage at or before use since they only need to reflect condition of previous instruction.
12. 3 Types of stalls: dependencies, resource scarcity, don’t know where to fetch. Stall is identified by front stage of stall (first stage that doesn’t move)

**ADD**

1. Fetch
2. Decode
3. Generate Address
   * SO.DMAR <- physical (GA.DLogical)
     1. SIB\_Base <- GPR(GA.Base)
     2. SIB\_Index <- GPR(GA.Index)
   * SO.SR\_dest <- GA.SR\_dest
   * SO.SR\_src <- GA.SR\_src
   * SO.imm <- GA.imm
   * Reg\_in\_use stall on:
     1. !GRP\_ready & GA.SIB\_needed
   * Dep stall
     1. (GA.Base == SO.DR) & GA.SIB\_needed & SO.V\_LD\_reg
     2. (GA.Base == E.DR) & GA.SIB\_needed & E.V\_LD\_reg
     3. (GA.Base == WB.DR) & GA.SIB\_needed & WB.V\_LD\_reg
     4. (GA.Index == SO.DR) & GA.SIB\_needed & SO.V\_LD\_reg
     5. (GA.Index == E.DR) & GA.SIB\_needed & E.V\_LD\_reg
     6. (GA.Index == WB.DR) & GA.SIB\_needed & WB.V\_LD\_reg
     7. (GA.Base == SO.DR2) & GA.SIB\_needed & SO.V\_LD\_reg2
     8. (GA.Base == E.DR2) & GA.SIB\_needed & E.V\_LD\_reg2
     9. (GA.Base == WB.DR2) & GA.SIB\_needed & WB.V\_LD\_reg2
     10. (GA.Index == SO.DR2) & GA.SIB\_needed & SO.V\_LD\_reg2
     11. (GA.Index == E.DR2) & GA.SIB\_needed & E.V\_LD\_reg2
     12. (GA.Index == WB.DR2) & GA.SIB\_needed & WB.V\_LD\_reg2
4. Setup Operands
   * E.op\_dest <- r/m
   * E.op\_src <- r/m, imm
   * Dep stall on:
     1. (SO.DR == E.DR) & SO.V\_CS\_DR\_needed & E.V\_CS\_LD\_reg
     2. (SO.DR == WB.DR) & SO.V\_CS\_DR\_needed & WB.V\_CS\_LD\_reg
     3. (SO.SR == E.DR) & SO.V\_CS\_SR \_needed & E.V\_ CS\_LD\_reg
     4. (SO.SR == WB.DR) & SO.V\_CS\_SR\_needed & WB.V\_ CS\_LD\_reg
     5. (physical (SO. DMAR) == E.DMAR) & SO.V\_CS\_Dmem\_needed & E.V\_Mem\_wb
     6. (physical (SO. DMAR) == WB.DMAR) & SO.V\_CS\_Dmem\_needed & WB.V\_Mem\_wb
   * Mem Stall on:
     1. (SO.CS\_Dcache.en & SO.V) & !DCache\_r\_priority1
5. Execute
   * WB.alu\_result <- E.op\_dest + E.op\_src
6. Write Back
   * r/m <- WB.alu\_result
   * flags <- WB.alu\_flags
   * Mem stall on:
     1. WB.CS\_Dcache.en & WB.V & !DCache\_r\_priority0

**CMOVC**

1. Fetch
2. Decode
3. Generate Address
   * SO.DMAR <- physical (GA.DLogical)
   * SO.SR\_dest <- GA.SR\_dest
   * SO.SR\_src <- GA.SR\_src
4. Setup Operands
   * E.op\_src <- r/m
   * Dep stall on:
     1. (SO.SR == E.DMAR) & SO.CS\_SR \_needed & E.V\_LD\_reg
     2. (SO.SR == WB.DMAR) & SO.CS\_SR\_needed & WB.V\_LD\_reg
     3. (physical (SO. DMAR) == E.DMAR) & SO.CS\_Dmem\_needed & E.V\_LD\_DMAR
     4. (physical (SO. DMAR) == WB.DMAR) & SO.CS\_Dmem\_needed & WB.V\_LD\_DMAR
   * Mem Stall on:
     1. SO.CS\_Dcache.en & SO.V & DCache\_r\_priority1
5. Execute
   * WB.alu\_result < - pass(E.op\_src)
6. Write Back
   * read(flags) -> r/m <-WB.alu\_result
   * Mem stall on:
     1. WB.CS\_Dcache.en & WB.V & DCache\_r\_priority0

**DAA**

1. Fetch
2. Decode
3. Generate Address
   * SO.SR\_dest <-GA.SR\_dest
   * SO.SR\_src <- GA.SR\_src
4. Setup Operands
   * E.op\_src <- r/m, imm
   * Dep stall on:
     1. (SO.SR\_src == E.SR\_src) & SO.CS\_SR\_dest\_needed & E.V\_LD\_reg
     2. (SO.SR\_src == WB.SR\_src) & SO.CS\_SR\_dest\_needed & WB.V\_LD\_reg
5. Execute
   * WB.alu\_result <- decimal\_adjust(E.op\_src)
6. Write Back
   * r/m <- WB.alu\_result
   * flags <- WB.alu\_flags

**HLT**

1. Fetch
2. Decode
3. Generate Address
4. Setup Operands
5. Execute
6. Write Back
   * Stall Forever

**JNE/JNBE** (no matter where we determine the target of the CC there will be a bubble)

1. Fetch
   * Branch stall on: (until I know target)
     1. DE.V\_BR\_STALL
     2. GA.V\_BR\_STALL
     3. SO.V\_BR\_STALL
2. Decode
3. Generate Address
   * Read(flags) -> EIP <- EIP + rel8/16/32
   * Dep stall:
     1. V\_SO\_LD\_CC & GA\_V
     2. V\_E\_LD\_CC & GA\_V
     3. V\_WB\_LD\_CC & GA\_V
4. Setup Operands
5. Execute
6. Write Back

**JMP**

1. Fetch
   * Branch stall on:
     1. DE.V\_BR\_STALL
     2. GA.V\_BR\_STALL
     3. SO.V\_BR\_STALL
2. Decode
3. Generate Address
   * SO.DMAR <- physical (GA.DLogical)
     1. SIB\_Base <- GPR(GA.Base)
     2. SIB\_Index <- GPR(GA.Index)
   * SO.SR\_dest <- GA.SR\_dest
   * SO.SR\_src <- GA.SR\_src
   * SO.imm <- GA.imm
   * Reg\_in\_use stall on:
     1. !GRP\_ready & GA.SIB\_needed
   * Dep stall
     1. (GA.Base == SO.DR) & GA.SIB\_needed & SO.V\_LD\_reg
     2. (GA.Base == E.DR) & GA.SIB\_needed & E.V\_LD\_reg
     3. (GA.Base == WB.DR) & GA.SIB\_needed & WB.V\_LD\_reg
     4. (GA.Index == SO.DR) & GA.SIB\_needed & SO.V\_LD\_reg
     5. (GA.Index == E.DR) & GA.SIB\_needed & E.V\_LD\_reg
     6. (GA.Index == WB.DR) & GA.SIB\_needed & WB.V\_LD\_reg
     7. (GA.Base == SO.DR2) & GA.SIB\_needed & SO.V\_LD\_reg2
     8. (GA.Base == E.DR2) & GA.SIB\_needed & E.V\_LD\_reg2
     9. (GA.Base == WB.DR2) & GA.SIB\_needed & WB.V\_LD\_reg2
     10. (GA.Index == SO.DR2) & GA.SIB\_needed & SO.V\_LD\_reg2
     11. (GA.Index == E.DR2) & GA.SIB\_needed & E.V\_LD\_reg2
     12. (GA.Index == WB.DR2) & GA.SIB\_needed & WB.V\_LD\_reg2
4. Setup Operands
   * E.op\_dest <- r/m
   * E.op\_src <- r/m, imm
   * Dep stall on:
     1. (SO.DR == E.DR) & SO.V\_CS\_DR\_needed & E.V\_CS\_LD\_reg
     2. (SO.DR == WB.DR) & SO.V\_CS\_DR\_needed & WB.V\_CS\_LD\_reg
     3. (SO.SR == E.DR) & SO.V\_CS\_SR \_needed & E.V\_ CS\_LD\_reg
     4. (SO.SR == WB.DR) & SO.V\_CS\_SR\_needed & WB.V\_ CS\_LD\_reg
     5. (physical (SO.DLogical) == E.DMAR) & SO.V\_CS\_Dmem\_needed & E.V\_Mem\_wb
     6. (physical (SO.DLogical) == WB.DMAR) & SO.V\_CS\_Dmem\_needed & WB.V\_Mem\_wb
   * Mem Stall on:
     1. (SO.CS\_Dcache.en & SO.V) & !DCache\_r\_priority1
5. Execute
   * EIP <- Addr(DE.Addresses)
6. Write Back

**MOV**

1. Fetch
2. Decode
3. Generate Address
   * SO.DMAR <- physical (GA.DLogical)
     1. SIB\_Base <- GPR(GA.Base)
     2. SIB\_Index <- GPR(GA.Index)
   * SO.SR\_dest <- GA.SR\_dest
   * SO.SR\_src <- GA.SR\_src
   * SO.imm <- GA.imm
   * Reg\_in\_use stall on:
     1. !GRP\_ready & GA.SIB\_needed
   * Dep stall
     1. (GA.Base == SO.DR) & GA.SIB\_needed & SO.V\_LD\_reg
     2. (GA.Base == E.DR) & GA.SIB\_needed & E.V\_LD\_reg
     3. (GA.Base == WB.DR) & GA.SIB\_needed & WB.V\_LD\_reg
     4. (GA.Index == SO.DR) & GA.SIB\_needed & SO.V\_LD\_reg
     5. (GA.Index == E.DR) & GA.SIB\_needed & E.V\_LD\_reg
     6. (GA.Index == WB.DR) & GA.SIB\_needed & WB.V\_LD\_reg
     7. (GA.Base == SO.DR2) & GA.SIB\_needed & SO.V\_LD\_reg2
     8. (GA.Base == E.DR2) & GA.SIB\_needed & E.V\_LD\_reg2
     9. (GA.Base == WB.DR2) & GA.SIB\_needed & WB.V\_LD\_reg2
     10. (GA.Index == SO.DR2) & GA.SIB\_needed & SO.V\_LD\_reg2
     11. (GA.Index == E.DR2) & GA.SIB\_needed & E.V\_LD\_reg2
     12. (GA.Index == WB.DR2) & GA.SIB\_needed & WB.V\_LD\_reg2
4. Setup Operands
   * E.op\_src <- r/m, imm
   * Dep stall on:
     1. (DE.SR == SO.DR) & DE.V\_CS\_SR\_needed & SO.V\_ CS\_LD\_reg
     2. (DE.SR == E.DR) & DE.V\_CS\_SR \_needed & E.V\_ CS\_LD\_reg
     3. (DE.SR == WB.DR) & DE.V\_CS\_SR\_needed & WB.V\_ CS\_LD\_reg
     4. (physical (DE.DLogical) == SO.DMAR) & DE.V\_CS\_Dmem\_needed & SO.V\_Mem\_wb
     5. (physical (DE.DLogical) == E.DMAR) & DE.V\_CS\_Dmem\_needed & E.V\_Mem\_wb
     6. (physical (DE.DLogical) == WB.DMAR) & DE.V\_CS\_Dmem\_needed & WB.V\_Mem\_wb
   * Mem Stall on:
     1. (SO.CS\_Dcache.en & SO.V) & !DCache\_r\_priority1
5. Execute
   * WB.alu\_result <- pass(E.op\_src)
6. Write Back
   * r/m <- WB.alu\_result
   * flags <- WB.alu\_flags
   * Mem stall on:
     1. WB.CS\_Dcache.en & WB.V & !DCache\_r\_priority0

**MOVQ**

1. Fetch
2. Decode
3. Generate Address
   * SO.DMAR <- physical (GA.DLogical)
     1. SIB\_Base <- GPR(GA.Base)
     2. SIB\_Index <- GPR(GA.Index)
   * SO.MM\_SR\_dest <- GA.MM\_SR\_dest
   * SO.MM\_SR\_src <- GA.MM\_SR\_src
   * Reg\_in\_use stall on:
     1. !GRP\_ready & DE.SIB\_needed
   * Dep stall
     1. (GA.Base == SO.DR) & GA.SIB\_needed & SO.V\_LD\_reg
     2. (GA.Base == E.DR) & GA.SIB\_needed & E.V\_LD\_reg
     3. (GA.Base == WB.DR) & GA.SIB\_needed & WB.V\_LD\_reg
     4. (GA.Index == SO.DR) & GA.SIB\_needed & SO.V\_LD\_reg
     5. (GA.Index == E.DR) & GA.SIB\_needed & E.V\_LD\_reg
     6. (GA.Index == WB.DR) & GA.SIB\_needed & WB.V\_LD\_reg
     7. (GA.Base == SO.DR2) & GA.SIB\_needed & SO.V\_LD\_reg2
     8. (GA.Base == E.DR2) & GA.SIB\_needed & E.V\_LD\_reg2
     9. (GA.Base == WB.DR2) & GA.SIB\_needed & WB.V\_LD\_reg2
     10. (GA.Index == SO.DR2) & GA.SIB\_needed & SO.V\_LD\_reg2
     11. (GA.Index == E.DR2) & GA.SIB\_needed & E.V\_LD\_reg2
     12. (GA.Index == WB.DR2) & GA.SIB\_needed & WB.V\_LD\_reg2
4. Setup Operands
   * E.MM\_op\_src <- r/m
   * Dep stall on:
     1. (SO. MM\_SR == E. MMDR) & SO.V\_CS\_MM\_SR \_needed & E.V\_ CS\_LD\_MMreg
     2. (SO. MM\_SR == WB. MMDR) & SO.V\_CS\_MM\_SR\_needed & WB.V\_CS\_LD\_MMreg
     3. (physical (SO.MM\_MAR) ==lap E.DMAR or E.MM\_MAR) & DE.V\_CS\_Dmem\_needed & E.V\_Mem\_wb
     4. (physical (SO.MM\_MAR) ==lap WB.DMAR or E.MM\_MAR) & DE.V\_CS\_Dmem\_needed & WB.V\_Mem\_wb
   * Mem Stall on:
     1. (SO.CS\_Dcache.en & SO.V) & !DCache\_r\_priority1
5. Execute
   * WB.MM\_alu\_result <- pass(E.MMop\_src)
6. Write Back
   * r/m <- WB.MM\_alu\_result
   * Mem stall on:
     1. WB.CS\_Dcache.en & WB.V & !DCache\_r\_priority0

**OR**

1. Fetch
2. Decode
3. Generate Address
   * SO.DMAR <- physical (GA.DLogical)
     1. SIB\_Base <- GPR(GA.Base)
     2. SIB\_Index <- GPR(GA.Index)
   * SO.SR\_dest <- GA.SR\_dest
   * SO.SR\_src <- GA.SR\_src
   * SO.imm <- GA.imm
   * Reg\_in\_use stall on:
     1. !GRP\_ready & GA.SIB\_needed
   * Dep stall
     1. (GA.Base == SO.DR) & GA.SIB\_needed & SO.V\_LD\_reg
     2. (GA.Base == E.DR) & GA.SIB\_needed & E.V\_LD\_reg
     3. (GA.Base == WB.DR) & GA.SIB\_needed & WB.V\_LD\_reg
     4. (GA.Index == SO.DR) & GA.SIB\_needed & SO.V\_LD\_reg
     5. (GA.Index == E.DR) & GA.SIB\_needed & E.V\_LD\_reg
     6. (GA.Index == WB.DR) & GA.SIB\_needed & WB.V\_LD\_reg
     7. (GA.Base == SO.DR2) & GA.SIB\_needed & SO.V\_LD\_reg2
     8. (GA.Base == E.DR2) & GA.SIB\_needed & E.V\_LD\_reg2
     9. (GA.Base == WB.DR2) & GA.SIB\_needed & WB.V\_LD\_reg2
     10. (GA.Index == SO.DR2) & GA.SIB\_needed & SO.V\_LD\_reg2
     11. (GA.Index == E.DR2) & GA.SIB\_needed & E.V\_LD\_reg2
     12. (GA.Index == WB.DR2) & GA.SIB\_needed & WB.V\_LD\_reg2
4. Setup Operands
   * E.op\_dest <- r/m
   * E.op\_src <- r/m, imm
   * Dep stall on:
     1. (SO.DR == E.DR) & SO.V\_CS\_DR\_needed & E.V\_CS\_LD\_reg
     2. (SO.DR == WB.DR) & SO.V\_CS\_DR\_needed & WB.V\_CS\_LD\_reg
     3. (SO.SR == E.DR) & SO.V\_CS\_SR \_needed & E.V\_ CS\_LD\_reg
     4. (SO.SR == WB.DR) & SO.V\_CS\_SR\_needed & WB.V\_ CS\_LD\_reg
     5. (physical (SO. DMAR) == E.DMAR) & DE.V\_CS\_Dmem\_needed & E.V\_Mem\_wb
     6. (physical (SO. DMAR) == WB.DMAR) & DE.V\_CS\_Dmem\_needed & WB.V\_Mem\_wb
   * Mem Stall on:
     1. (SO.CS\_Dcache.en & SO.V) & !DCache\_r\_priority1
5. Execute
   * WB.alu\_result <- E.op\_dest | E.op\_src
6. Write Back
   * r/m <- WB.alu\_result
   * flags <- WB.alu\_flags
   * Mem stall on:
     1. WB.CS\_Dcache.en & WB.V & !DCache\_r\_priority0

**PADDD**

1. Fetch
2. Decode
3. Generate Address
   * SO.DMAR <- physical (GA.DLogical)
     1. SIB\_Base <- GPR(GA.Base)
     2. SIB\_Index <- GPR(GA.Index)
   * SO.MMSR\_dest <- GA.MMSR\_dest
   * SO.MMSR\_src <- GA.MMSR\_src
   * Reg\_in\_use stall on:
     1. !GRP\_ready & GA.SIB\_needed
   * Dep stall
     1. (GA.Base == SO.DR) & GA.SIB\_needed & SO.V\_LD\_reg
     2. (GA.Base == E.DR) & GA.SIB\_needed & E.V\_LD\_reg
     3. (GA.Base == WB.DR) & GA.SIB\_needed & WB.V\_LD\_reg
     4. (GA.Index == SO.DR) & GA.SIB\_needed & SO.V\_LD\_reg
     5. (GA.Index == E.DR) & GA.SIB\_needed & E.V\_LD\_reg
     6. (GA.Index == WB.DR) & GA.SIB\_needed & WB.V\_LD\_reg
     7. (GA.Base == SO.DR2) & GA.SIB\_needed & SO.V\_LD\_reg2
     8. (GA.Base == E.DR2) & GA.SIB\_needed & E.V\_LD\_reg2
     9. (GA.Base == WB.DR2) & GA.SIB\_needed & WB.V\_LD\_reg2
     10. (GA.Index == SO.DR2) & GA.SIB\_needed & SO.V\_LD\_reg2
     11. (GA.Index == E.DR2) & GA.SIB\_needed & E.V\_LD\_reg2
     12. (GA.Index == WB.DR2) & GA.SIB\_needed & WB.V\_LD\_reg2
4. Setup Operands
   * E.MMop\_src <- r/m
   * E.MMop\_dest <- r
   * Dep stall on:
     1. (DE. MMSR == SO. MMDR) & DE.V\_CS\_MMSR\_needed & SO.V\_ CS\_LD\_MMreg
     2. (DE. MMSR == E. MMDR) & DE.V\_CS\_MMSR \_needed & E.V\_ CS\_LD\_MMreg
     3. (DE. MMSR == WB. MMDR) & DE.V\_CS\_MMSR\_needed & WB.V\_CS\_LD\_MMreg
     4. (DE. MMDR == SO. MMDR) & DE.V\_CS\_MMDR\_needed & SO.V\_ CS\_LD\_MMreg
     5. (DE. MMDR == E. MMDR) & DE.V\_CS\_MMDR \_needed & E.V\_ CS\_LD\_MMreg
     6. (DE. MMDR == WB. MMDR) & DE.V\_CS\_MMDR\_needed & WB.V\_CS\_LD\_MMreg
     7. (SO. MMSR == E. MMDR) & SO.V\_CS\_MMSR \_needed & E.V\_ CS\_LD\_MMreg
     8. (SO. MMSR == WB. MMDR) & SO.V\_CS\_MMSR\_needed & WB.V\_CS\_LD\_MMreg
     9. (physical (SO.MM\_MAR) ==lap E.DMAR or E.MM\_MAR) & DE.V\_CS\_Dmem\_needed & E.V\_Mem\_wb
     10. (physical (SO.MM\_MAR) ==lap WB.DMAR or E.MM\_MAR) & DE.V\_CS\_Dmem\_needed & WB.V\_Mem\_wb
   * Mem Stall on:
     1. (SO.CS\_Dcache.en & SO.V) & !DCache\_r\_priority1
5. Execute
   * WB.MMalu\_result <- pass(E.MMop\_src + E.MMop\_dest)
6. Write Back
   * r/m <- WB.MMalu\_result
   * Mem stall on:
     1. WB.CS\_Dcache.en & WB.V & !DCache\_r\_priority0

**POP**

1. Fetch
2. Decode
3. Generate Address
   * SO.DMAR <- physical (GA.DLogical)
     1. SIB\_Base <- GPR(GA.Base)
     2. SIB\_Index <- GPR(GA.Index)
   * SO.Stack\_MAR <- Top\_Of\_Stack\_Reg
   * If(valid & stack increment) -> Top\_Of\_Stack\_Reg + GA.SIZE\_ATTRIBUTE
   * SO.SR\_dest <- GA.SR\_dest
   * Reg\_in\_use stall on:
     1. !GRP\_ready & GA.SIB\_needed
   * Dep stall
     1. (GA.Base == SO.DR) & GA.SIB\_needed & SO.V\_LD\_reg
     2. (GA.Base == E.DR) & GA.SIB\_needed & E.V\_LD\_reg
     3. (GA.Base == WB.DR) & GA.SIB\_needed & WB.V\_LD\_reg
     4. (GA.Index == SO.DR) & GA.SIB\_needed & SO.V\_LD\_reg
     5. (GA.Index == E.DR) & GA.SIB\_needed & E.V\_LD\_reg
     6. (GA.Index == WB.DR) & GA.SIB\_needed & WB.V\_LD\_reg
4. Setup Operand
   * E.op\_dest <- m (stack)
   * E.op\_src <- r/m
   * Dep stall on:
     1. (SO.SR == E.DR) & SO.V\_CS\_SR \_needed & E.V\_ CS\_LD\_reg
     2. (SO.SR == WB.DR) & SO.V\_CS\_SR\_needed & WB.V\_ CS\_LD\_reg
     3. SO.STACK & E.V\_CS\_Stack\_needed & SO.V\_Stack
     4. SO.STACK & WB.V\_CS\_ Stack \_needed & SO.V\_Stack
     5. (SO.DMAR == E.DMAR) & SO.V\_CS\_Dmem\_needed & E.V\_Mem\_wb
     6. (SO.DMAR == WB.DMAR) & SO.V\_CS\_Dmem\_needed & WB.V\_Mem\_wb
   * Mem Stall on:
     1. (SO.CS\_Dcache.en & SO.V) & !DCache\_r\_priority1
5. Execute
   * WB.alu\_result <- pass(E.op\_src)
6. Write Back
   * r/m <- WB.alu\_result
   * Mem stall on:
     1. WB.CS\_Dcache.en & WB.V & !DCache\_r\_priority0

**PSHUFW**

1. Fetch
2. Decode
3. Generate Address
   * SO.DMAR <- physical (GA.DLogical)
     1. SIB\_Base <- GPR(GA.Base)
     2. SIB\_Index <- GPR(GA.Index)
   * SO.MMSR\_dest <- GA.MMSR\_dest
   * SO.MMSR\_src <- GA.MMSR\_src
   * SO.imm <- GA.imm
   * Reg\_in\_use stall on:
     1. !GRP\_ready & DE.SIB\_needed
   * Dep stall
     1. (GA.Base == SO.DR) & GA.SIB\_needed & SO.V\_LD\_reg
     2. (GA.Base == E.DR) & GA.SIB\_needed & E.V\_LD\_reg
     3. (GA.Base == WB.DR) & GA.SIB\_needed & WB.V\_LD\_reg
     4. (GA.Index == SO.DR) & GA.SIB\_needed & SO.V\_LD\_reg
     5. (GA.Index == E.DR) & GA.SIB\_needed & E.V\_LD\_reg
     6. (GA.Index == WB.DR) & GA.SIB\_needed & WB.V\_LD\_reg
4. Setup Operands
   * E.MM\_op\_src <- r/m
   * E.MM\_op\_dest <- r/m
   * E.imm <- SO.imm
   * Dep stall on:
     1. (SO. MMSR == E. MMDR) & SO.V\_CS\_MMSR \_needed & E.V\_ CS\_LD\_MMreg
     2. (SO. MMSR == WB. MMDR) & SO.V\_CS\_MMSR\_needed & WB.V\_CS\_LD\_MMreg
     3. (physical (SO.MM\_MAR) ==lap E.DMAR or E.MM\_MAR) & DE.V\_CS\_Dmem\_needed & E.V\_Mem\_wb
     4. (physical (SO.MM\_MAR) ==lap WB.DMAR or E.MM\_MAR) & DE.V\_CS\_Dmem\_needed & WB.V\_Mem\_wb
   * Mem Stall on:
     1. (SO.CS\_Dcache.en & SO.V) & !DCache\_r\_priority1
5. Execute
   * WB.MM\_alu\_result <- shuffle (E.MMop\_src, E.imm)
6. Write Back
   * r/m <- WB.alu\_result
   * Mem stall on:
     1. WB.CS\_Dcache.en & WB.V & !DCache\_r\_priority0

**SAL/SAR**

1. Fetch
2. Decode
3. Generate Address
   * SO.DMAR <- physical (GA.DLogical)
     1. SIB\_Base <- GPR(GA.Base)
     2. SIB\_Index <- GPR(GA.Index)
   * SO.SR\_dest <- GA.SR\_dest
   * SO.SR\_src <- GA.SR\_src
   * SO.imm <- GA.imm
   * Reg\_in\_use stall on:
     1. !GRP\_ready & GA.SIB\_needed
   * Dep stall
     1. (GA.Base == SO.DR) & GA.SIB\_needed & SO.V\_LD\_reg
     2. (GA.Base == E.DR) & GA.SIB\_needed & E.V\_LD\_reg
     3. (GA.Base == WB.DR) & GA.SIB\_needed & WB.V\_LD\_reg
     4. (GA.Index == SO.DR) & GA.SIB\_needed & SO.V\_LD\_reg
     5. (GA.Index == E.DR) & GA.SIB\_needed & E.V\_LD\_reg
     6. (GA.Index == WB.DR) & GA.SIB\_needed & WB.V\_LD\_reg
     7. (GA.Base == SO.DR2) & GA.SIB\_needed & SO.V\_LD\_reg2
     8. (GA.Base == E.DR2) & GA.SIB\_needed & E.V\_LD\_reg2
     9. (GA.Base == WB.DR2) & GA.SIB\_needed & WB.V\_LD\_reg2
     10. (GA.Index == SO.DR2) & GA.SIB\_needed & SO.V\_LD\_reg2
     11. (GA.Index == E.DR2) & GA.SIB\_needed & E.V\_LD\_reg2
     12. (GA.Index == WB.DR2) & GA.SIB\_needed & WB.V\_LD\_reg2
4. Setup Operands
   * E.op\_dest <- r/m
   * E.op\_src <- r/imm
   * Dep stall on:
     1. (SO.DR == E.DR) & SO.V\_CS\_DR\_needed & E.V\_CS\_LD\_reg
     2. (SO.DR == WB.DR) & SO.V\_CS\_DR\_needed & WB.V\_CS\_LD\_reg
     3. (SO.SR == E.DR) & SO.V\_CS\_SR \_needed & E.V\_ CS\_LD\_reg
     4. (SO.SR == WB.DR) & SO.V\_CS\_SR\_needed & WB.V\_ CS\_LD\_reg
     5. (physical (SO. DMAR) == E.DMAR) & SO.V\_CS\_Dmem\_needed & E.V\_Mem\_wb
     6. (physical (SO. DMAR) == WB.DMAR) & SO.V\_CS\_Dmem\_needed & WB.V\_Mem\_wb
   * Mem Stall on:
     1. (SO.CS\_Dcache.en & SO.V) & !DCache\_r\_priority1
5. Execute
   * WB.alu\_result <- E.op\_dest << or >> E.op\_src
6. Write Back
   * r/m <- WB.alu\_result
   * flags <- WB.alu\_flags
   * Mem stall on:
     1. WB.CS\_Dcache.en & WB.V & !DCache\_r\_priority0

**XCHG**

1. Fetch
2. Decode
3. Generate Address
   * SO.DMAR <- physical (GA.DLogical)
     1. SIB\_Base <- GPR(GA.Base)
     2. SIB\_Index <- GPR(GA.Index)
   * SO.SR\_dest <- GA.SR\_dest
   * SO.SR\_src <- GA.SR\_src
   * Reg\_in\_use stall on:
     1. !GRP\_ready & GA.SIB\_needed
   * Dep stall
     1. (GA.Base == SO.DR) & GA.SIB\_needed & SO.V\_LD\_reg
     2. (GA.Base == E.DR) & GA.SIB\_needed & E.V\_LD\_reg
     3. (GA.Base == WB.DR) & GA.SIB\_needed & WB.V\_LD\_reg
     4. (GA.Index == SO.DR) & GA.SIB\_needed & SO.V\_LD\_reg
     5. (GA.Index == E.DR) & GA.SIB\_needed & E.V\_LD\_reg
     6. (GA.Index == WB.DR) & GA.SIB\_needed & WB.V\_LD\_reg
     7. (GA.Base == SO.DR2) & GA.SIB\_needed & SO.V\_LD\_reg2
     8. (GA.Base == E.DR2) & GA.SIB\_needed & E.V\_LD\_reg2
     9. (GA.Base == WB.DR2) & GA.SIB\_needed & WB.V\_LD\_reg2
     10. (GA.Index == SO.DR2) & GA.SIB\_needed & SO.V\_LD\_reg2
     11. (GA.Index == E.DR2) & GA.SIB\_needed & E.V\_LD\_reg2
     12. (GA.Index == WB.DR2) & GA.SIB\_needed & WB.V\_LD\_reg2
4. Setup Operands
   * E.op\_dest <- r/m
   * E.op\_src <- r/m
   * Dep stall on:
     1. (SO.SR == E.DR) & SO.V\_CS\_SR \_needed & E.V\_ CS\_LD\_reg
     2. (SO.SR == WB.DR) & SO.V\_CS\_SR\_needed & WB.V\_ CS\_LD\_reg
     3. (SO.DR == E.DR2) & SO.V\_CS\_DR \_needed & E.V\_ CS\_LD\_reg2
     4. (SO.DR == WB.DR2) & SO.V\_DR\_SR\_needed & WB.V\_ CS\_LD\_reg2
     5. (physical (SO.DMAR) == E.DMAR) & DE.V\_CS\_Dmem\_needed & E.V\_Mem\_wb
     6. (physical (SO.DMAR) == WB.DMAR) & DE.V\_CS\_Dmem\_needed & WB.V\_Mem\_wb
   * Mem Stall on:
     1. (SO.CS\_Dcache.en & SO.V) & !DCache\_r\_priority1
5. Execute
   * WB.alu\_result <- pass(E.op\_src)
   * WB.DR2 <- E.op\_dest;
6. Write Back
   * r/m <- WB.alu\_result
   * r/m <- WB. DR2
   * Mem stall on:
     1. WB.CS\_Dcache.en & WB.V & !DCache\_r\_priority0

**POP**

1. Fetch
2. Decode
3. Generate Address
   * SO.DMAR <- physical (GA.DLogical)
     1. SIB\_Base <- GPR(GA.Base)
     2. SIB\_Index <- GPR(GA.Index)
   * SO.Stack\_MAR <- Top\_Of\_Stack\_Reg
   * If(valid & stack increment) -> Top\_Of\_Stack\_Reg - GA.SIZE\_ATTRIBUTE
   * SO.SR\_dest <- GA.SR\_dest
   * Reg\_in\_use stall on:
     1. !GRP\_ready & GA.SIB\_needed
   * Dep stall
     1. (GA.Base == SO.DR) & GA.SIB\_needed & SO.V\_LD\_reg
     2. (GA.Base == E.DR) & GA.SIB\_needed & E.V\_LD\_reg
     3. (GA.Base == WB.DR) & GA.SIB\_needed & WB.V\_LD\_reg
     4. (GA.Index == SO.DR) & GA.SIB\_needed & SO.V\_LD\_reg
     5. (GA.Index == E.DR) & GA.SIB\_needed & E.V\_LD\_reg
     6. (GA.Index == WB.DR) & GA.SIB\_needed & WB.V\_LD\_reg
     7. (GA.Base == SO.DR2) & GA.SIB\_needed & SO.V\_LD\_reg2
     8. (GA.Base == E.DR2) & GA.SIB\_needed & E.V\_LD\_reg2
     9. (GA.Base == WB.DR2) & GA.SIB\_needed & WB.V\_LD\_reg2
     10. (GA.Index == SO.DR2) & GA.SIB\_needed & SO.V\_LD\_reg2
     11. (GA.Index == E.DR2) & GA.SIB\_needed & E.V\_LD\_reg2
     12. (GA.Index == WB.DR2) & GA.SIB\_needed & WB.V\_LD\_reg2
4. Setup Operand
   * E.op\_dest <- m (stack)
   * E.op\_src <- r/m
   * Dep stall on:
     1. (SO.SR == E.DR) & SO.V\_CS\_SR \_needed & E.V\_ CS\_LD\_reg
     2. (SO.SR == WB.DR) & SO.V\_CS\_SR\_needed & WB.V\_ CS\_LD\_reg
     3. SO.STACK & E.V\_CS\_Stack\_needed & SO.V\_Stack
     4. SO.STACK & WB.V\_CS\_ Stack \_needed & SO.V\_Stack
     5. (SO.DMAR == E.DMAR) & SO.V\_CS\_Dmem\_needed & E.V\_Mem\_wb
     6. (SO.DMAR == WB.DMAR) & SO.V\_CS\_Dmem\_needed & WB.V\_Mem\_wb
   * Mem Stall on:
     1. (SO.CS\_Dcache.en & SO.V) & !DCache\_r\_priority1
5. Execute
   * WB.alu\_result <- pass(E.op\_src)
6. Write Back
   * r/m <- WB.alu\_result
   * Mem stall on:
     1. WB.CS\_Dcache.en & WB.V & !DCache\_r\_priority0