PIPELINED PROCESSOR

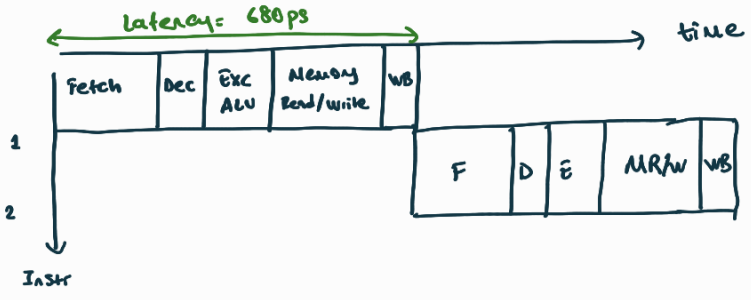
Reference: Ece Göran Schmidt's EE446 Slides from Harris

- Built top on the single cycle processor
- Add registers for each stage

1. **Fetch Stage**: Read instruction from instruction memory
2. **Decode Stage**: Read the source operands, decode instruction
3. **Execute Stage**: Perform ALU operation
4. **Memory stage**: Read/Write data memory
5. **Writeback stage**: Write the result to the register file

If # of stages <5
⇓
single-cycle processor

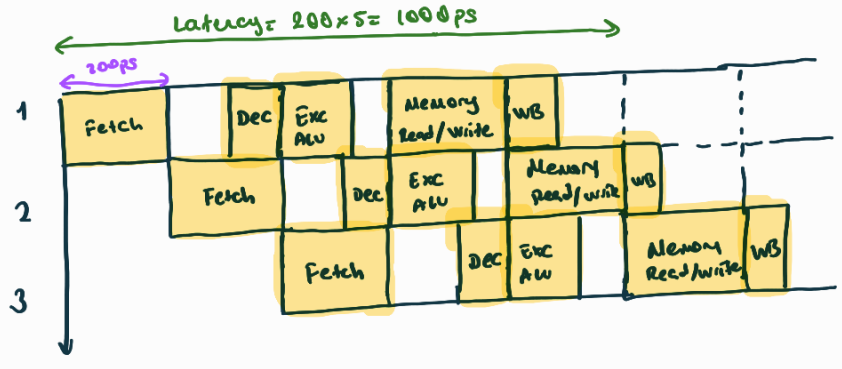
Critical Path and Logic Delays of Single Cycle



$T_{c_1} = t_{pcq\text{-}PC}$ (ignore) $+ 2t_{MEM} + t_{DEC} + t_{RF\,read} + t_{ALU} + 2t_{MUX} + t_{RF\,setup}$

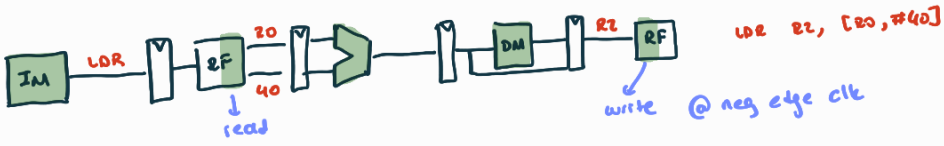
$T_{c_1} \simeq 680$ ps

Throughput: 1 instruction per 680 ps $\simeq$ 1.47 billion instruction

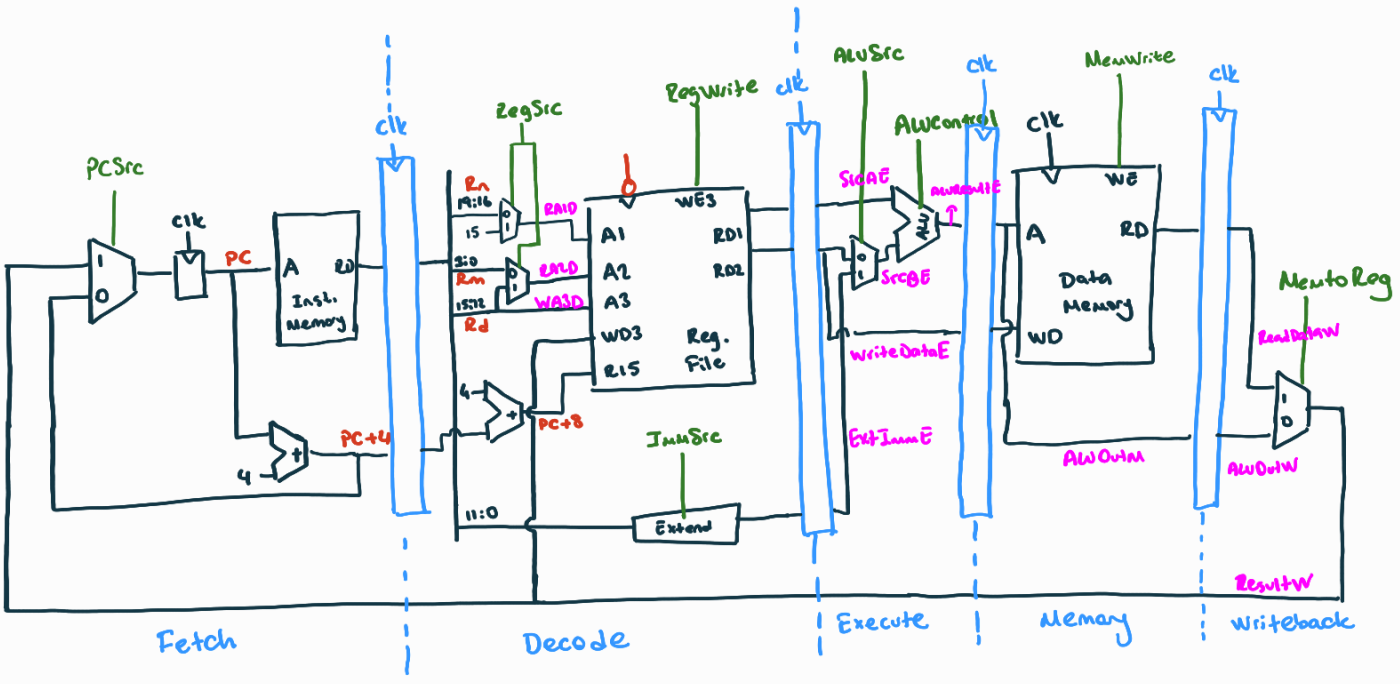


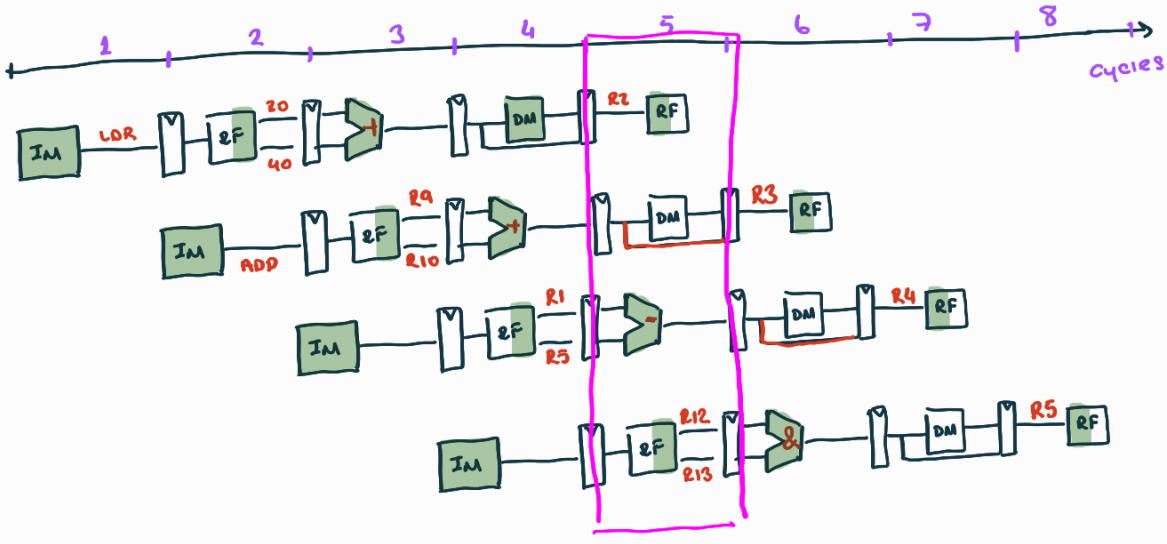
Throughput: 1 instruction per 200 ps $\simeq$ 5 billion instructions per second

Pipelined Processor Abstraction



Datapath Version 1





LDR R2, [R0, #40]

ADD R3, R9, R10

SUB R4, R1, R5

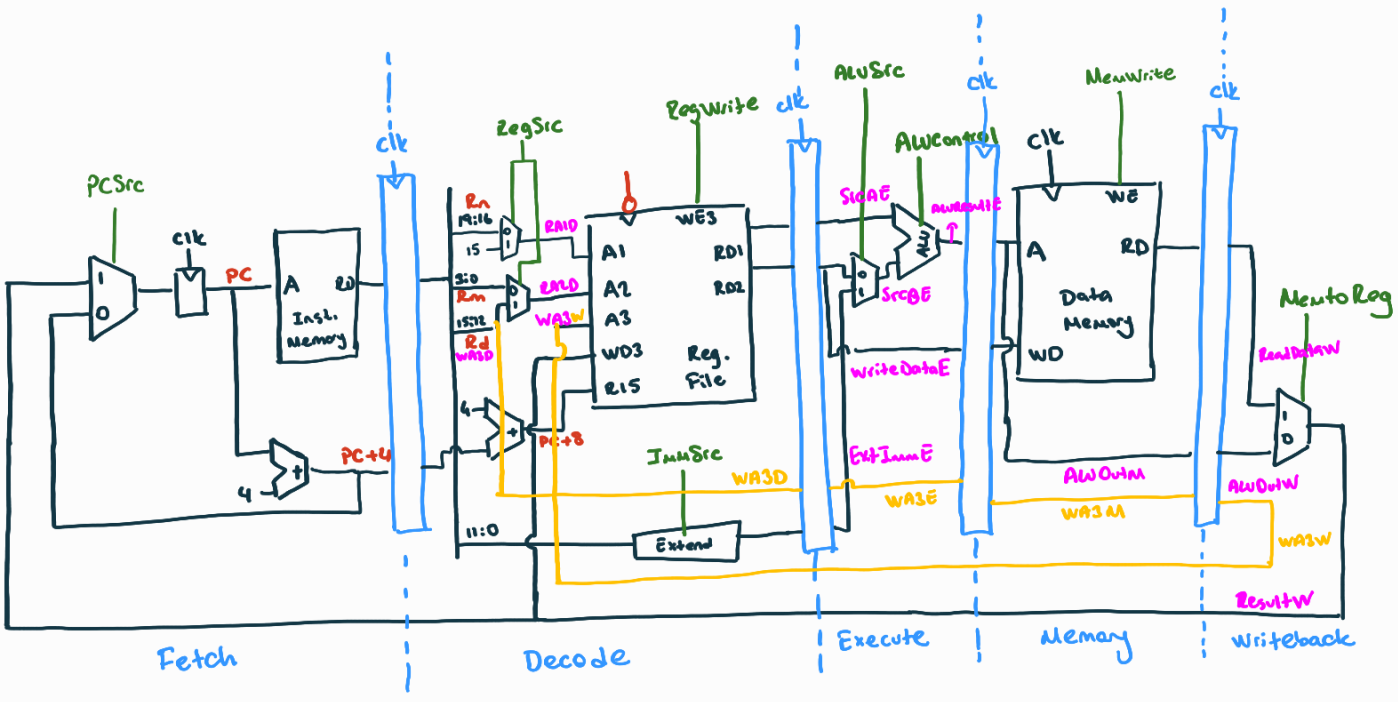
AND R5, R12, R13

! Problem 1: In Cycle 5, LDR is in WB → we want WA3D = R2

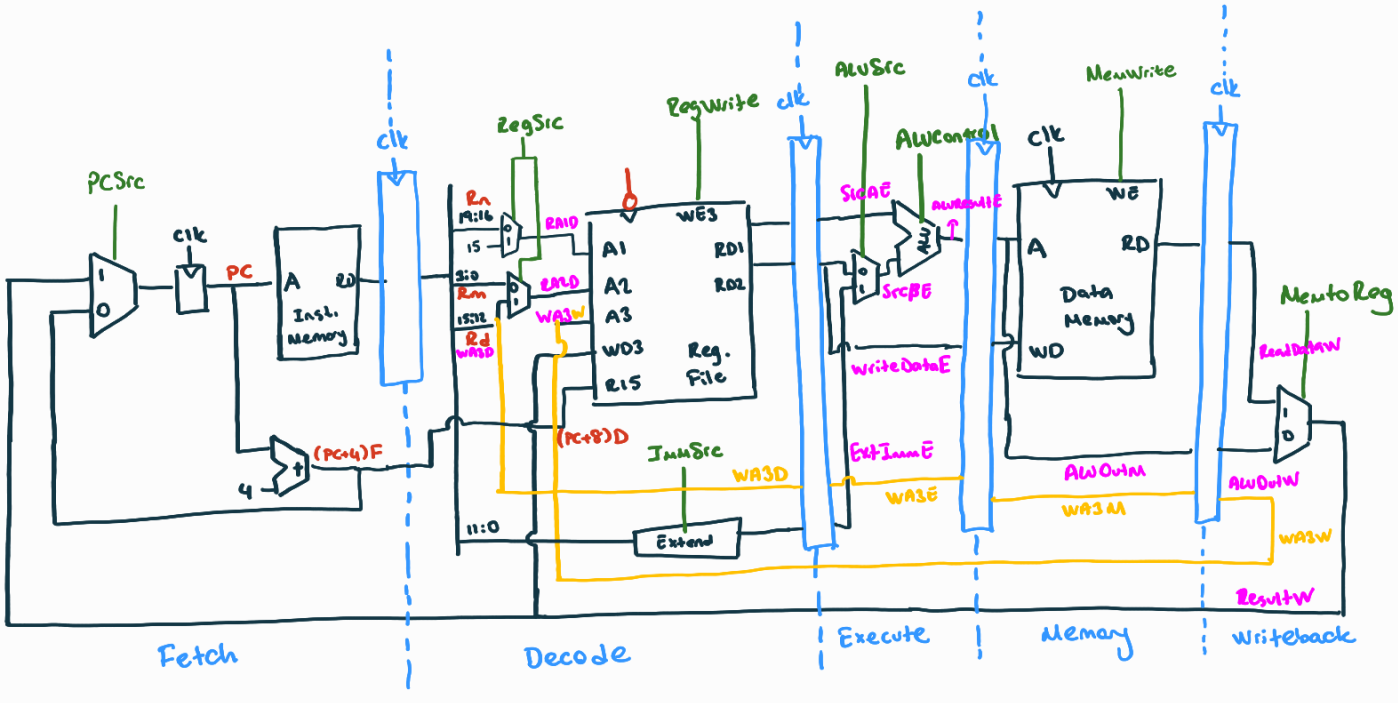
However AND is in decode WA3D = R5 !!

Rule: All signals associated with a particular instruction must advance through the pipeline in unison.

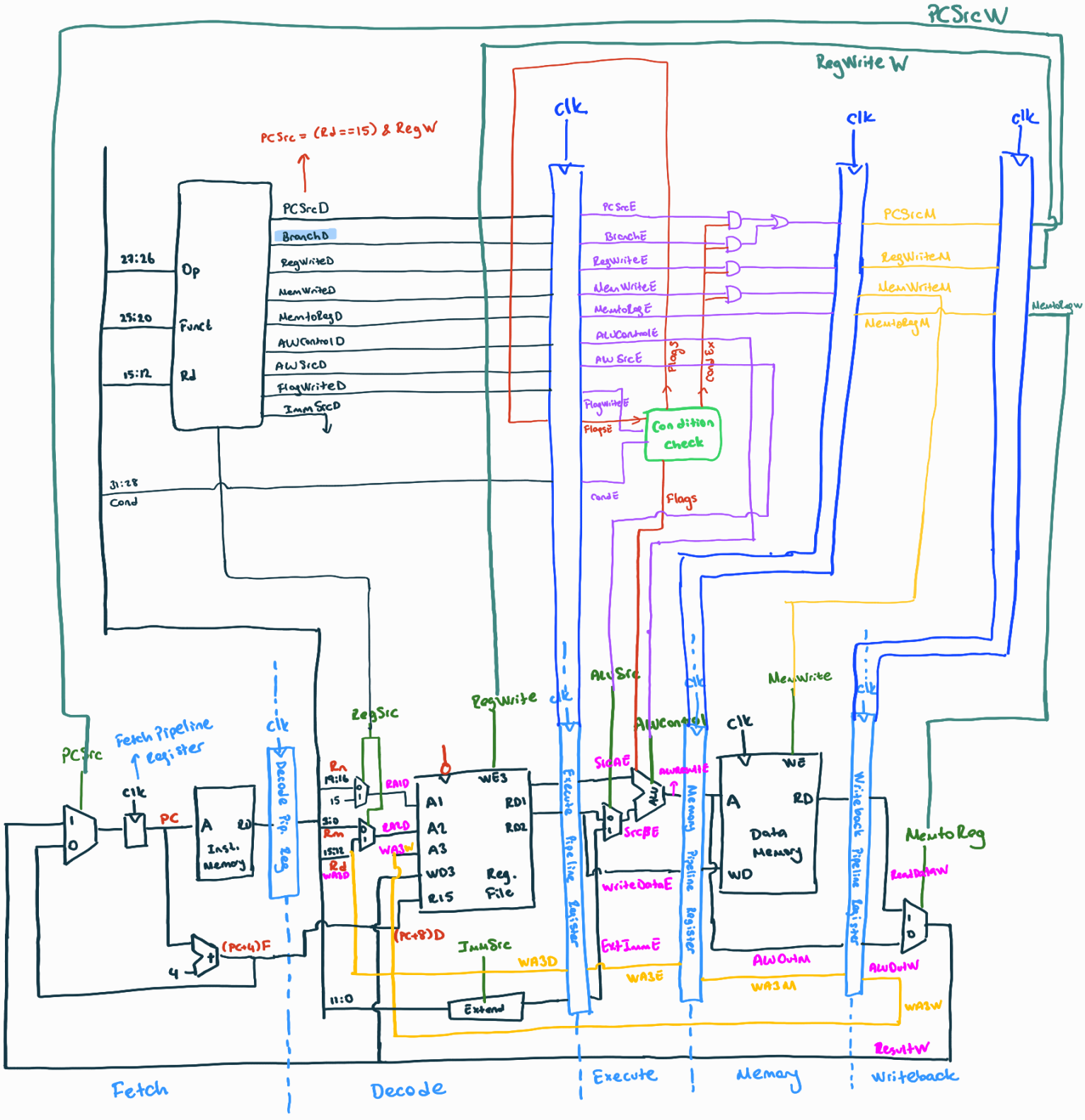
Datapath Version 2



Datapath Final Version 3



* At Fetch stage, logically (PC+4)F = PC+8. Thus there is no need for the additional adder.



* Branch signal is different from single-cycle.

Instruction Execution Example

0x A000 LDR R2, R0, #40

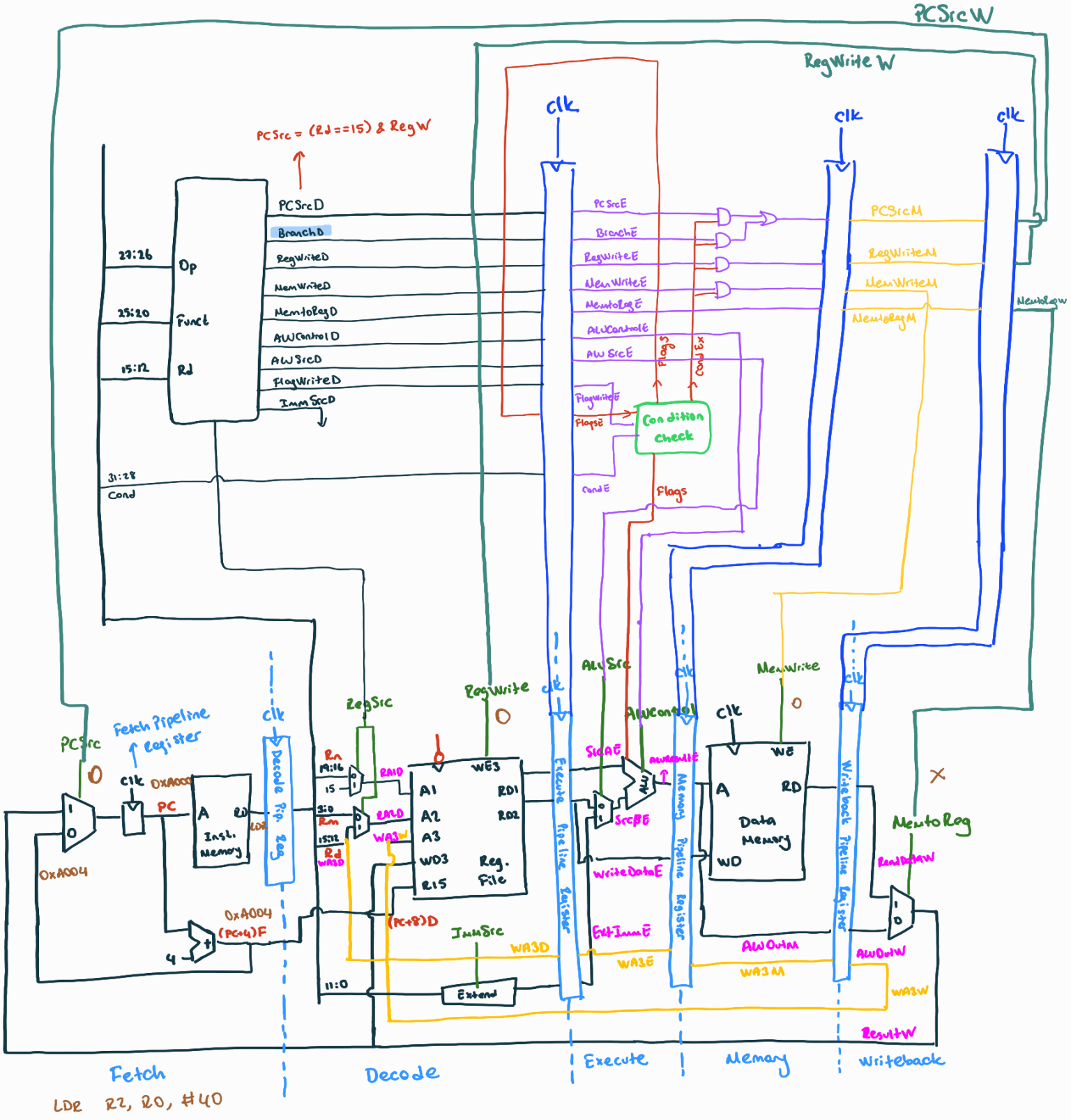
0x A004 AND R3, R5, R10

0x A008 STR R4, R1, #20

0x A00C BNI #10

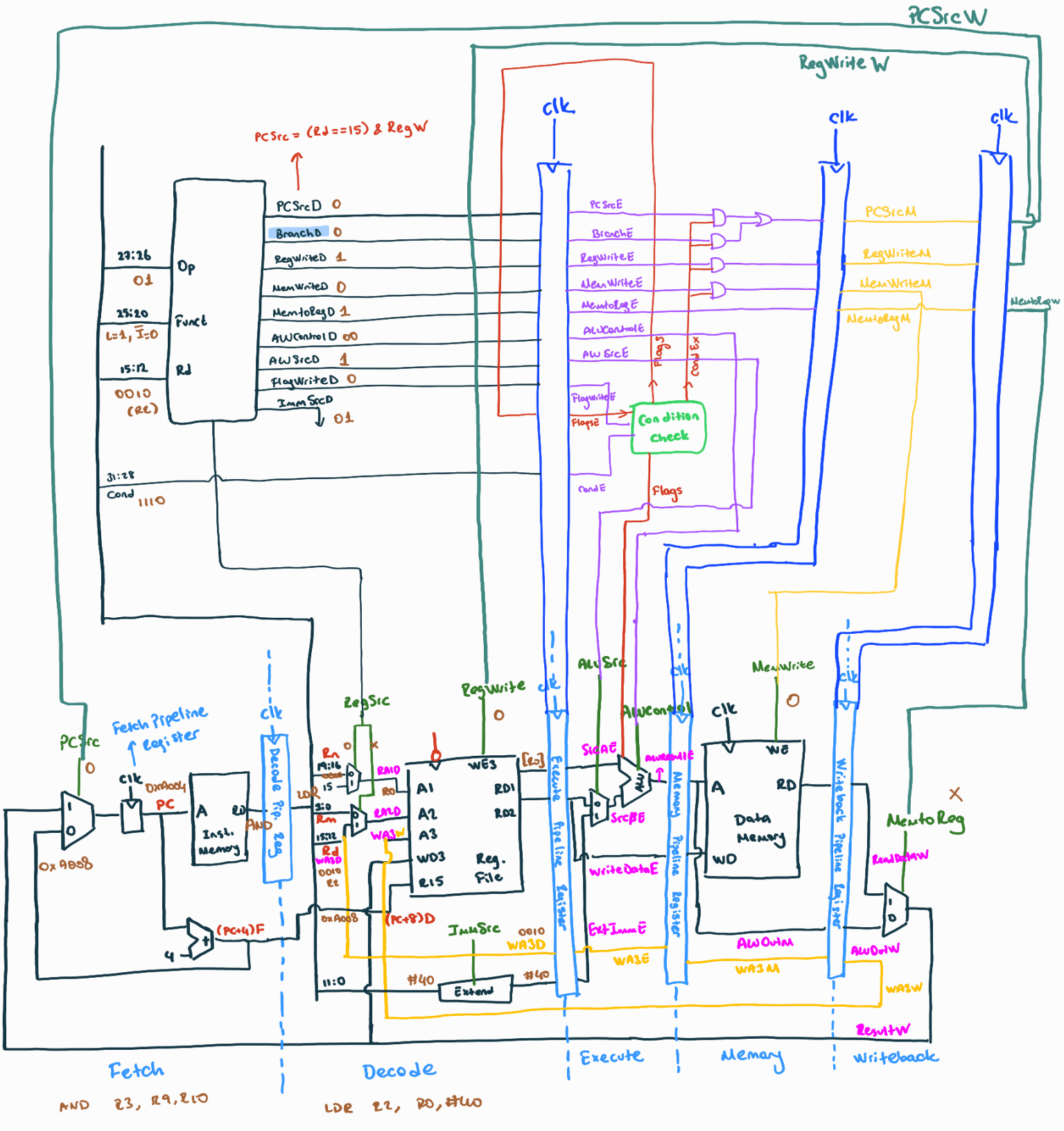
PCSrcW = 0, RegWriteW = 0, MemtoRegW = X, MemWriteM = 0

Cycle 1:

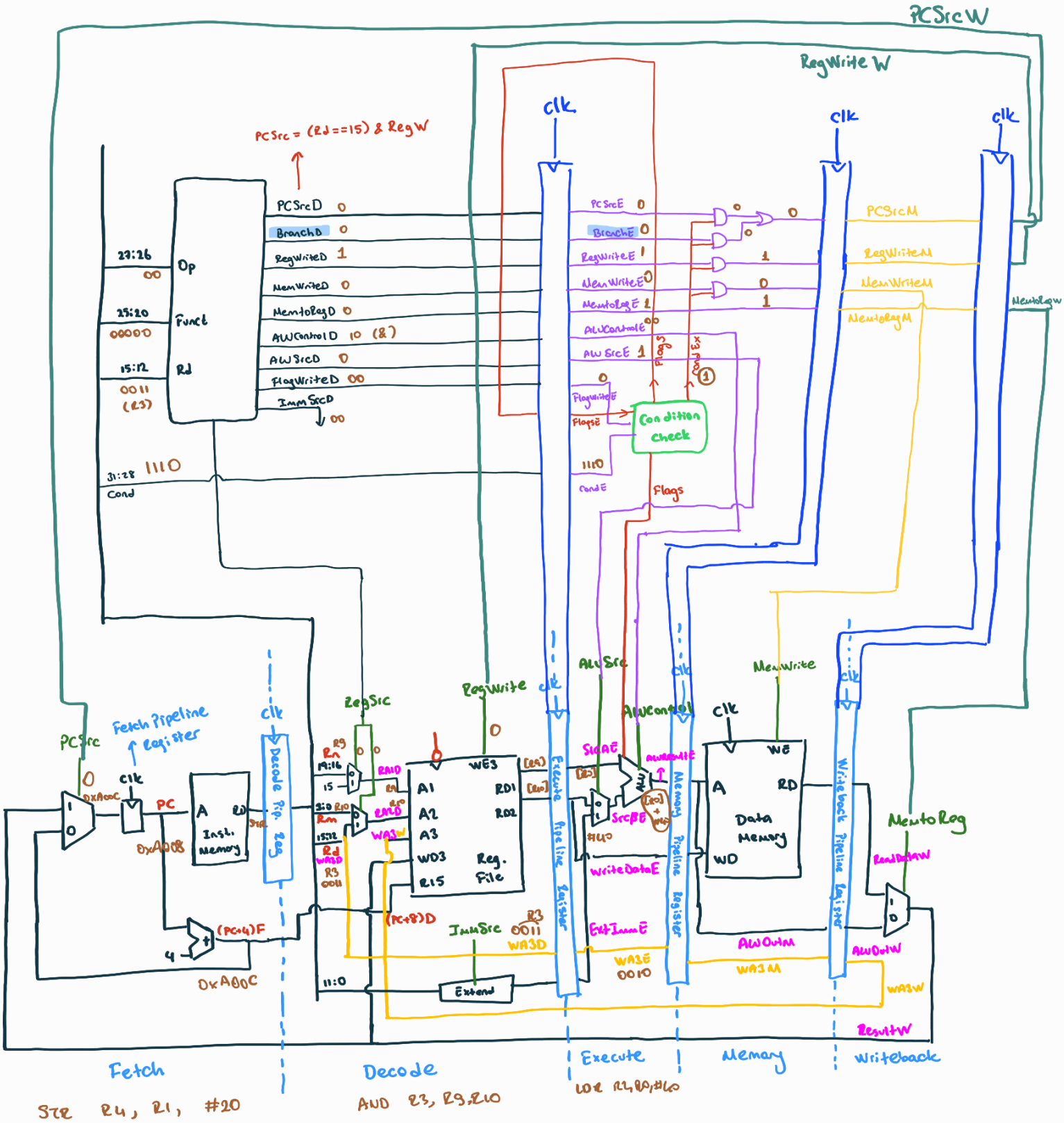


LDR R2, R0, #40

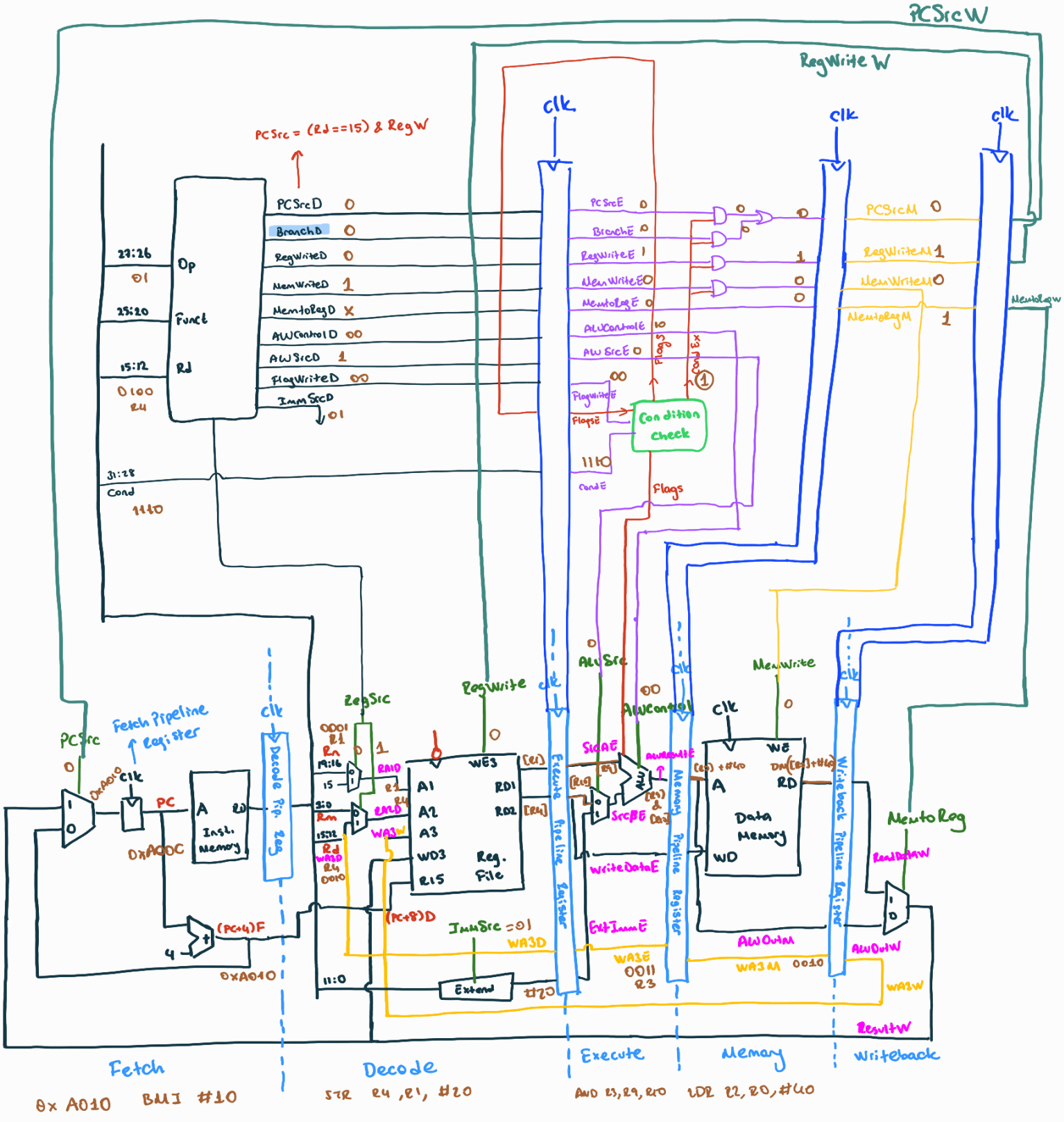
Cycle 2:


PCSrcW
RegWriteW
clk
PCSrc = (Rd==15) & RegW
PCSrcD 0
BranchD 0
RegWriteD 1
MemWriteD 0
MemtoRegD 1
ALUControlD 00
ALUSrcD 1
FlagWriteD 0
ImmSrcD 01
27:26
01
Op
25:20
L=1, I=0
Funct
15:12
Rd
0010
(R2)
31:28
Cond 1110
PCSrcE
BranchE
RegWriteE
MemWriteE
MemtoRegE
ALUControlE
ALUSrcE
FlagWriteE
FlagsE
CondEx
Flags
CondE
Condition check
PCSrcM
RegWriteM
MemWriteM
MemtoRegM
MemtoRegW
PCSrc
Fetch Pipeline register
clk
0x4008
PC
0xA004
Inst. Memory
A
RD
Decode Pipe. Reg
AND
(PC+4)F
4
RegSrc
Rn 19:16
15
Rm 3:0
Rd 15:12
WA3D 0010 R2
RA1D
RA2D
WA3W
RD
LDR
0xA008
(PC+8)D
RegWrite
WE3
A1
A2
A3
WD3
R15
RD1
RD2
Reg. File
[R2]
11:0
#40
Extend
ImmSrc
0010
WA3D
Execute pipeline register
ALUSrc
SrcAE
SrcBE
ALUControl
ALUOutE
WriteDataE
ExtImmE
WA3E
Memory Pipeline Register
MemWrite
WE
RD
Data Memory
WD
ALUOutM
WA3M
Writeback Pipeline Register
MemtoReg
ReadDataW
ALUOutW
WA3W
ResultW
Fetch
Decode
Execute
Memory
Writeback
AND R3, R9, R10
LDR R2, R0, #40

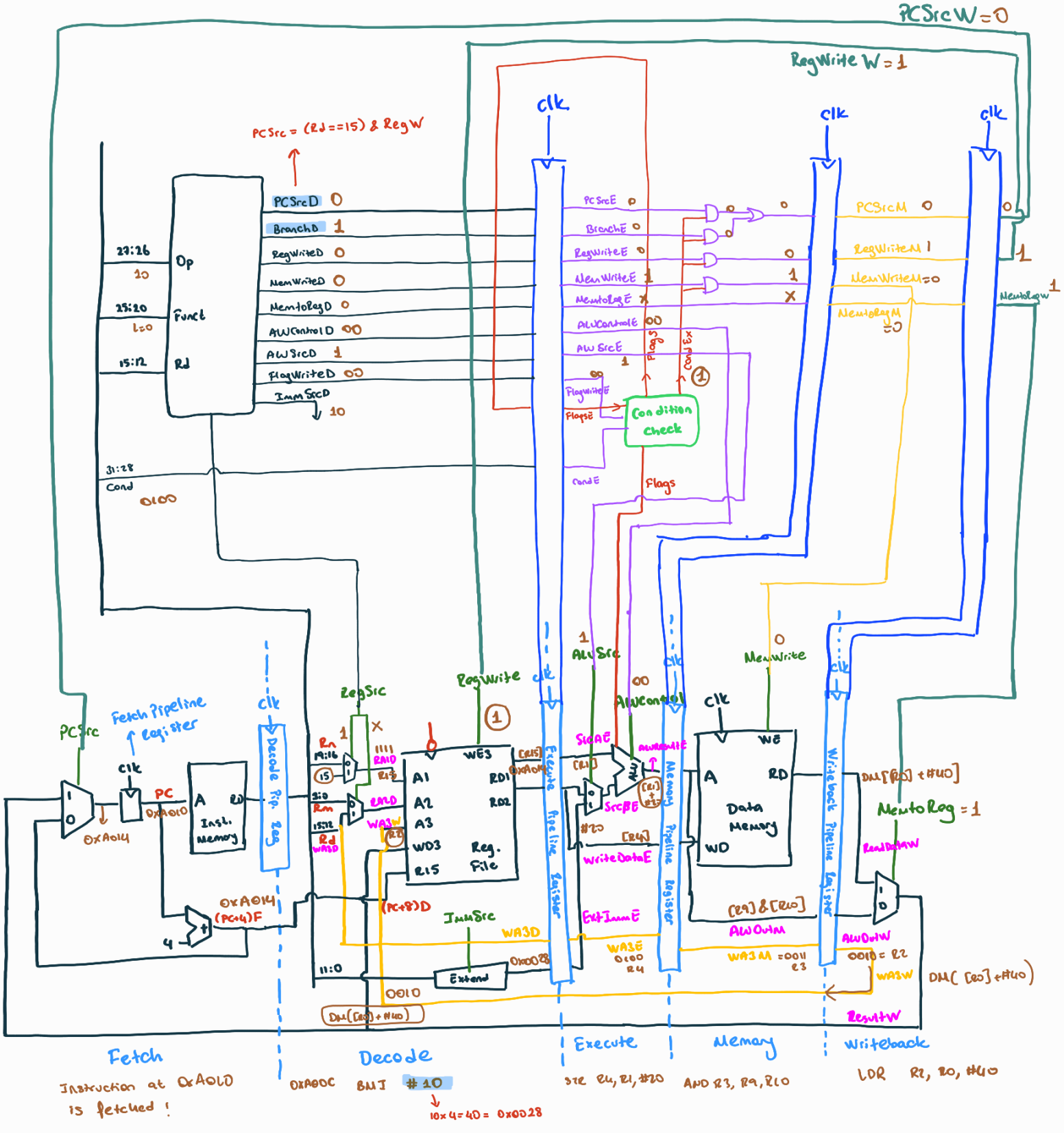
Cycle 3 :


PCSrcW
RegWriteW
clk
PCSrc = (Rd==15) & RegW
PCSrcD
BranchD
RegWriteD
MemWriteD
MemtoRegD
ALUControlD
ALUSrcD
FlagWriteD
ImmSrcD
23:26
Op
25:20
Funct
15:12
Rd
31:28
Cond
1110
PCSrcE
BranchE
RegWriteE
MemWriteE
MemtoRegE
ALUControlE
ALUSrcE
FlagWriteE
FlagsE
CondE
Condition check
Flags
PCSrcM
RegWriteM
MemWriteM
MemtoRegM
MemtoRegW
PCSrc
Fetch Pipeline register
PC
Instr. Memory
(PC+4)F
(PC+8)D
RegSrc
RegWrite
ImmSrc
ALUSrc
ALUControl
MemWrite
MemtoReg
Reg. File
Extend
Data Memory
SrcAE
SrcBE
WriteDataE
ExtImmE
WA3E
WA3M
WA3W
ALUOutM
ALUOutW
ReadDataW
ResultW
Fetch
Decode
Execute
Memory
Writeback
STR R4, R1, #20
AND R3, R9, R10
LDR R2, R0, #60

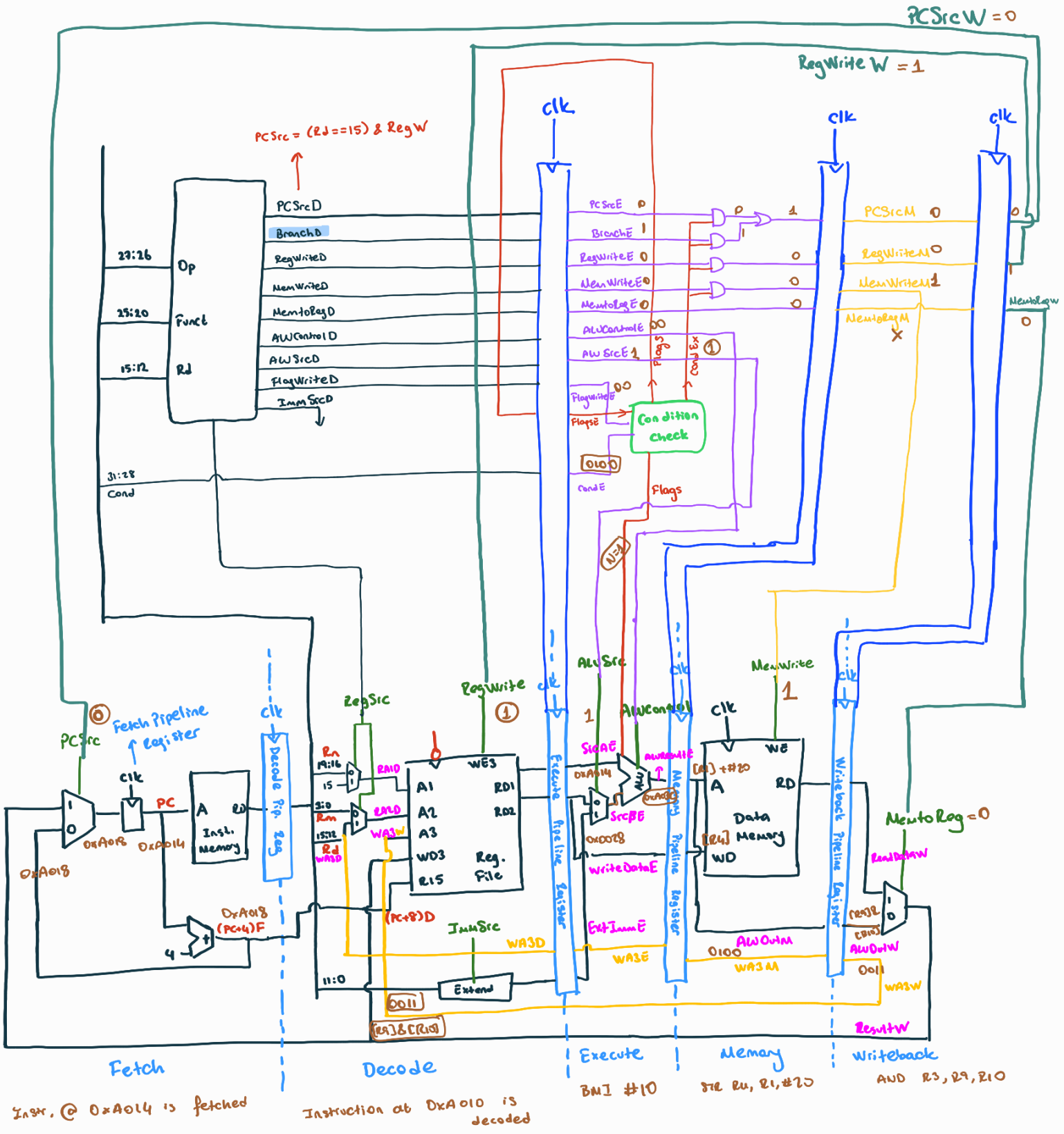
Cycle 4 :


PCSrcW
RegWriteW
clk
PCSrc = (Rd==15) & RegW
PCSrcD 0
BranchD 0
RegWriteD 0
MemWriteD 1
MemtoRegD X
ALUControlD 00
ALUSrcD 1
FlagWriteD 00
ImmSrcD 01
27:26
01
Op
25:20
Funct
15:12
Rd
0100
R4
31:28
Cond
1110
PCSrcE 0
BranchE 0
RegWriteE 1
MemWriteE 0
MemtoRegE 0
ALUControlE 10
ALUSrcE 0
FlagWriteE 00
FlagsE
Condition Check
CondE
Flags
4:3
PCSrcM 0
RegWriteM 1
MemWriteM 0
MemtoRegM 1
MemtoRegW
PCSrc
Fetch Pipeline register
0xA010
PC
0xA00C
Instr. Memory
(PC+4)F
4
Decode Pipeline Register
RegSrc
0001
R1
Rn
19:16
15
3:0
Rm
15:12
Rd
WA3D
R4
0010
RA1D
RA2D
WA3W
RegWrite
WE3
A1
A2
A3
WD3
R15
RD1
RD2
Reg. File
(PC+8)D
ImmSrc =01
WA3D
11:0
Extend
#20
ALUSrc
00
ALUControl
SrcAE
SrcBE
WriteDataE
ExtImmE
WA3E
0011
R3
MemWrite
0
clk
A
RD
Data Memory
WD
WE
ALUOutM
WA3M
0010
MemtoReg
ALUOutW
WA3W
ResultW
Execute pipeline register
Memory Pipeline Register
Writeback Pipeline Register
Fetch
Decode
Execute
Memory
Writeback
0x A010 BAL #10
STR R4, R1, #20
AND R3, R9, R0
LDR R2, R0, #40

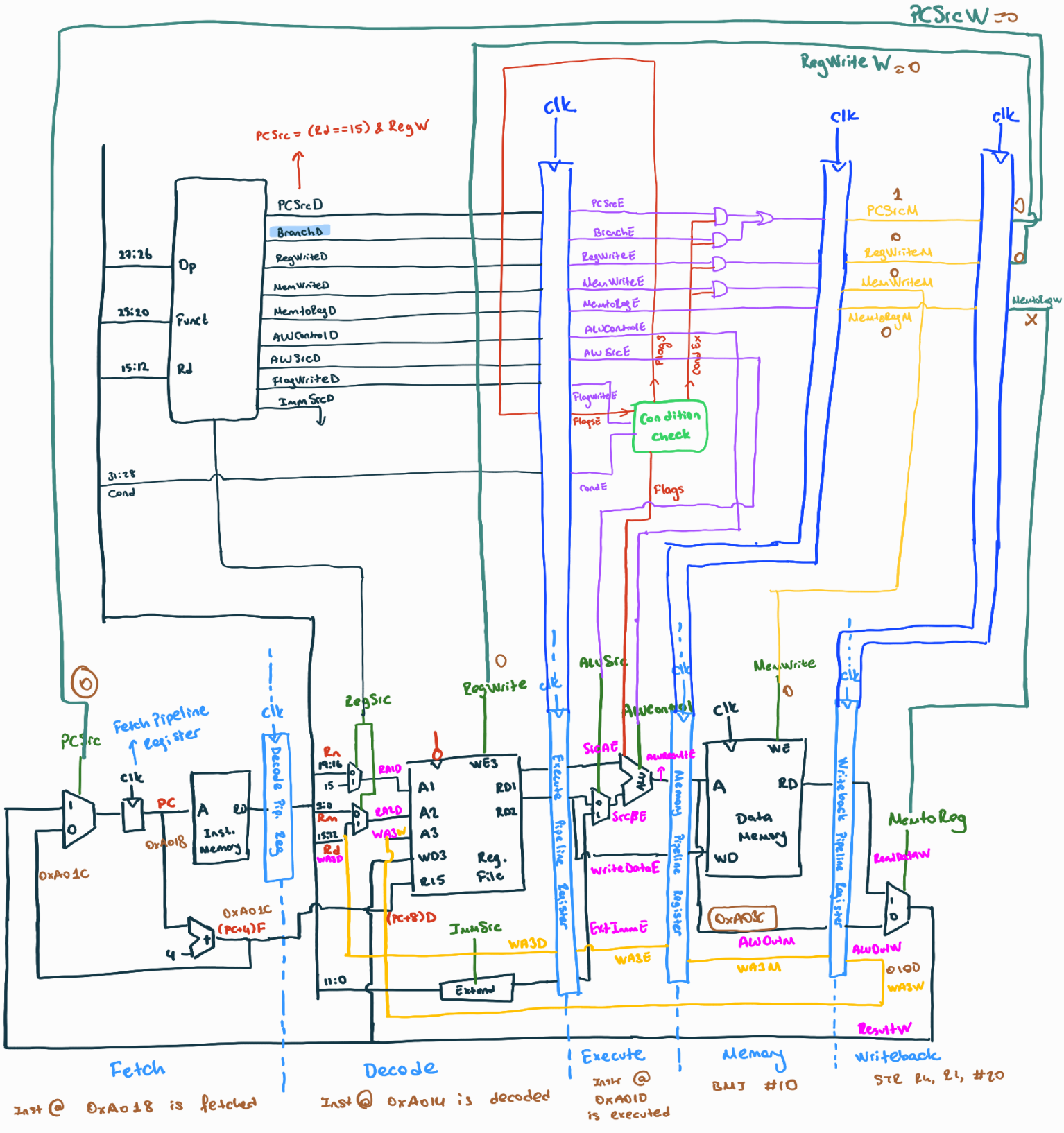
Cycle 5:


PCSrcW = 0
RegWrite W = 1
PCSrc = (RJ == 15) & RegW
clk
Op
Funct
Rd
PCSrcD 0
BranchD 1
RegWriteD 0
MemWriteD 0
MemtoRegD 0
ALUControlD 00
ALUSrcD 1
FlagWriteD 00
ImmSrcD 10
Cond 0100
PCSrcE
BranchE
RegWriteE
MemWriteE 1
MemtoRegE X
ALUControlE 00
ALUSrcE 1
FlagWriteE 00
Flags
CondEx
Condition check
CondE
PCSrcM 0
RegWriteM 1
MemWriteM = 0
MemtoRegM = 0
MemtoRegW 1
PCSrc
Fetch Pipeline register
PC
0xA010
0xA014
Instr. Memory
Decode Pipeline Reg
RegSrc
RegWrite
Reg. File
WE3
A1
A2
A3
WD3
R15
RD1
RD2
Extend
ImmSrc
(PC+8)D
WA3D
WA3E
WA3M = 0011 R3
WA3W
ALUSrc
ALUControl
SrcAE
SrcBE
#20
[R4]
writeDataE
ExtImmE
Execute pipeline register
Memory Pipeline Register
MemWrite
Data Memory
[R9] & [R10]
ALUOutM
Writeback Pipeline Register
MemtoReg = 1
ReadDataW
ALUOutW
0010 = R2
WA3W
ResultW
DM[R0] + #40
Fetch
Instruction at 0xA010 is fetched!
Decode
0xA00C BNI #10
10 x 4 = 40 = 0x0028
Execute
STR R4, R1, #20
Memory
AND R3, R9, R10
Writeback
LDR R2, R0, #40

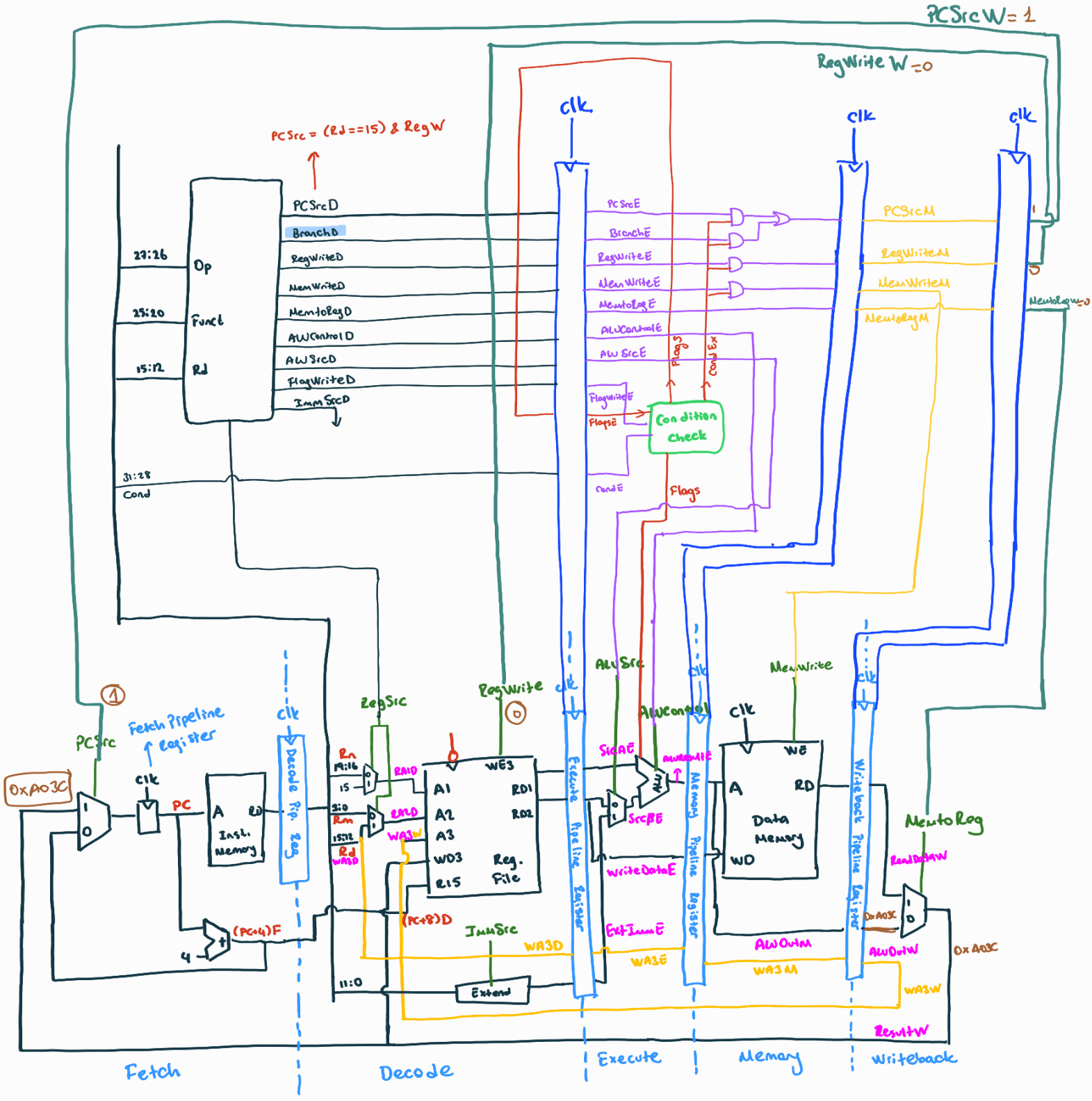
Cycle 6:


PCSrcW = 0
RegWrite W = 1
PCSrc = (Rd==15) & RegW
PCSrcD
BranchD
RegWriteD
MemWriteD
MemtoRegD
ALUControlD
ALUSrcD
FlagWriteD
ImmSrcD
23:26
25:20
15:12
31:28
Cond
Op
Funct
Rd
clk
PCSrcE
BranchE
RegWriteE
MemWriteE
MemtoRegE
ALUControlE
ALUSrcE
FlagWriteE
FlagsE
CondE
Condition check
Flags
PCSrcM
RegWriteM
MemWriteM
MemtoRegM
MemtoRegW
PCSrc
Fetch Pipeline register
PC
Inst. Memory
0xA018
0xA014
0xA018 (PC+4)F
Decode pipe reg
RegSrc
Rn
Rm
Rd
RA1D
RA2D
WA3W
WA3D
WE3
A1
A2
A3
WD3
R15
RD1
RD2
Reg. File
(PC+8)D
ImmSrc
Extend
11:0
0011
[R9]&[R10]
RegWrite
ALUSrc
ALUControl
SrcAE
SrcBE
0x0018
writeDataE
ExtImmE
WA3E
WA3M
ALUOutM
0100
MemWrite
1
Data Memory
[R4] +#20
WE
RD
WD
MemtoReg=0
ReadDataW
ALUOutW
WA3W
ResultW
Execute pipeline register
Memory pipeline register
Writeback Pipeline register
Fetch
Decode
Execute
Memory
Writeback
Instr. @ 0xA014 is fetched
Instruction at 0xA010 is decoded
BNI #10
STR R4, R1, #20
AND R3, R9, R10

Cycle 7:


PCSrcW = 0
RegWrite W = 0
PCSrc = (Rd == 15) & RegW
clk
PCSrcD
BranchD
RegWriteD
MemWriteD
MemtoRegD
ALUControlD
ALUSrcD
FlagWriteD
ImmSrcD
Op
Funct
Rd
31:28
Cond
27:26
25:20
15:12
PCSrcE
BranchE
RegWriteE
MemWriteE
MemtoRegE
ALUControlE
ALUSrcE
FlagWriteE
FlagsE
CondE
Flags
CondEx
Condition check
PCSrcM
RegWriteM
MemWriteM
MemtoRegM
MemtoRegW
PCSrc
Fetch Pipeline register
PC
Instr. Memory
0xA018
0xA01C
(PC+4)F
Decode Pipeline Register
RegSrc
Rn
Rd
RA1D
RA2D
WA3W
WA3D
(PC+8)D
RegWrite
WE3
A1
A2
A3
WD3
R15
RD1
RD2
Reg. File
ImmSrc
Extend
11:0
Execute Pipeline Register
ALUSrc
ALUControl
SrcAE
SrcBE
ALU
writeDataE
ExtImmE
WA3E
MemWrite
WE
A
RD
Data Memory
WD
0xA03C
ALUOutM
WA3M
Memory Pipeline Register
Writeback Pipeline Register
MemtoReg
ReadDataW
ALUOutW
WA3W
ResultW
Fetch
Inst @ 0xA018 is fetched
Decode
Inst @ 0xA01C is decoded
Execute
Inst @ 0xA01D is executed
Memory
BMI #10
Writeback
STR R4, R1, #20

Cycle 8:



Control Hazard — which instruction to fetch next?

HAZARD TYPES

Data Hazard

An instruction tries to read a register that has not yet been written back by a prev instruction

Control Hazard

The decision of what instruction to fetch next has not been made by the time the fetch takes place.

Data Hazard: Example 1

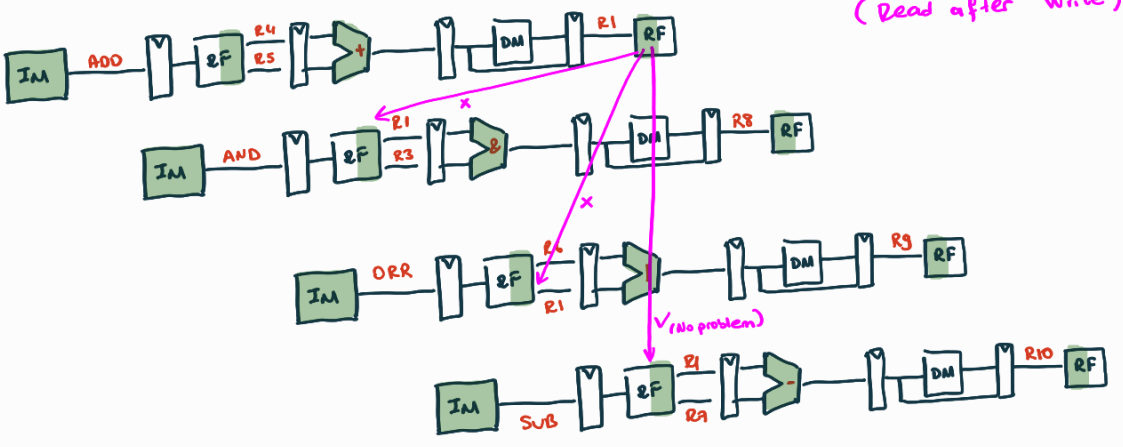
RAW Hazards
(Read after Write)

ADD R1, R4, R5

AND R8, R1, R3

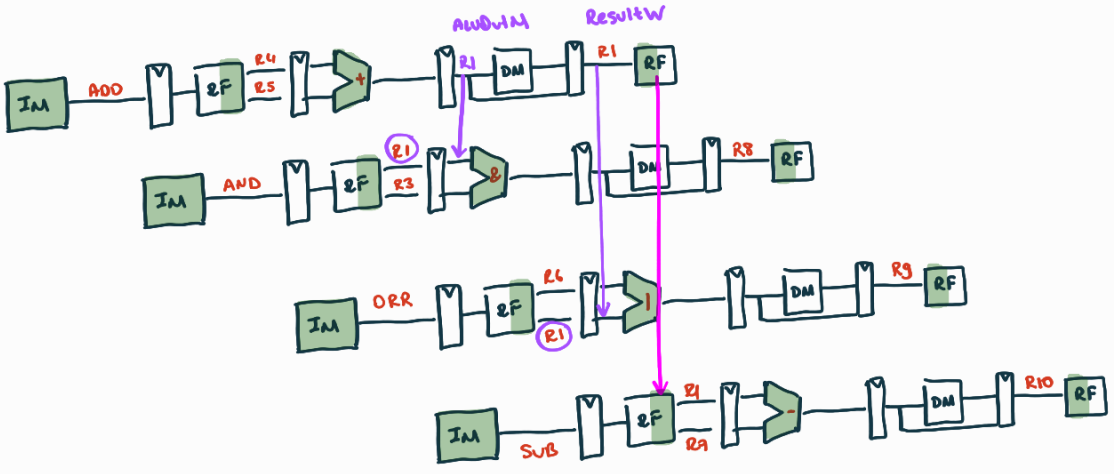
ORR R9, R6, R1

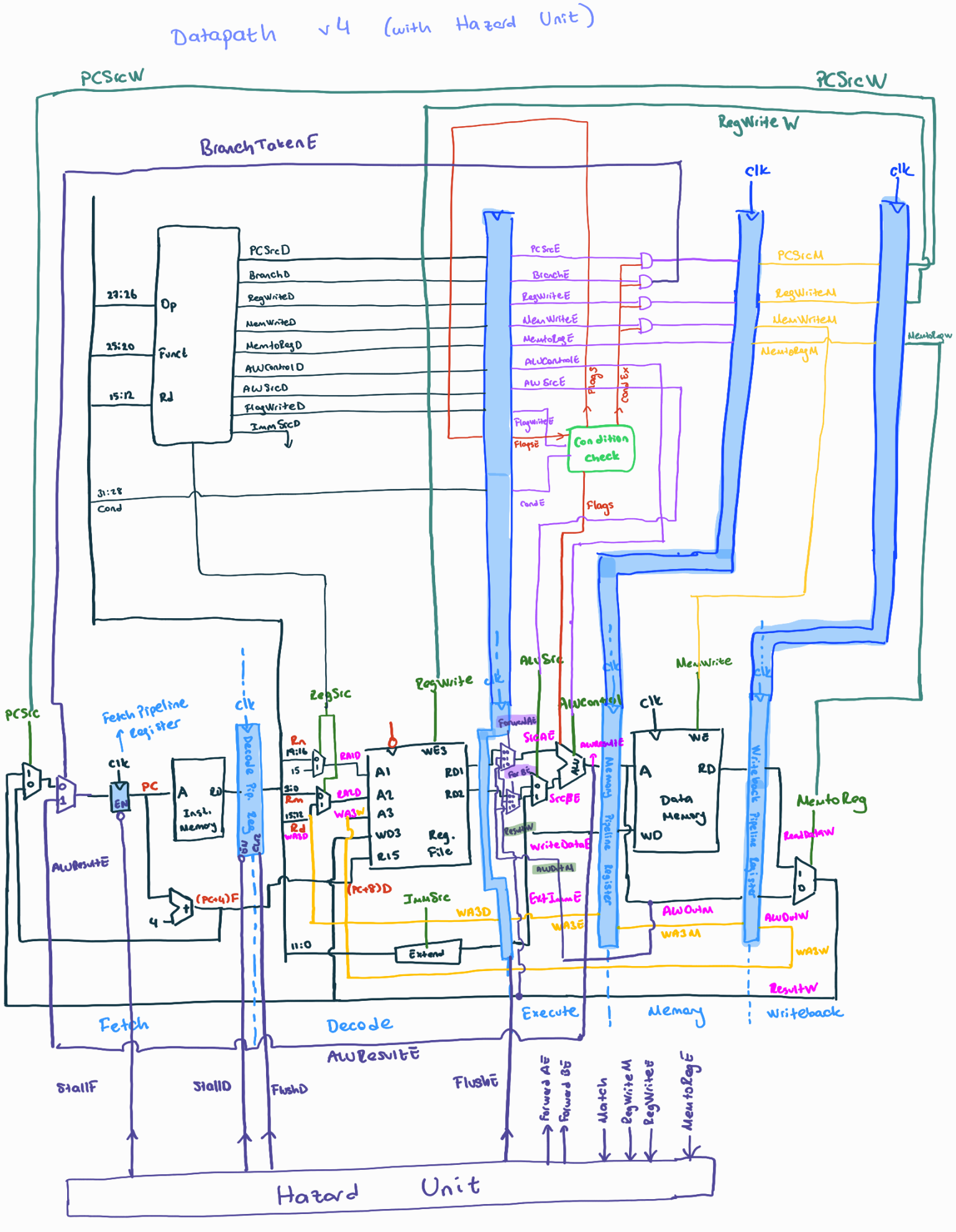
SUB R10, R1, R7



→ Software Solution: Adding 2 NOPs

→ Solution2: Forwarding




Datapath v4 (with Hazard Unit)
PCSrcW
PCSrcW
RegWriteW
BranchTakenE
clk
clk
PCSrcD
BranchD
RegWriteD
MemWriteD
MemtoRegD
ALUControlD
ALUSrcD
FlagWriteD
ImmSrcD
Op
Funct
Rd
25:26
25:20
15:12
31:28
Cond
PCSrcE
BranchE
RegWriteE
MemWriteE
MemtoRegE
ALUControlE
ALUSrcE
FlagWriteE
FlagsE
CondEx
Flags
CondE
Condition Check
PCSrcM
RegWriteM
MemWriteM
MemtoRegM
MemtoRegW
PCSrc
Fetch Pipeline register
clk
PC
A
RD
Instr. Memory
EN
ALUResultE
(PC+4)F
4
+
Decode pipeline register
RegSrc
Rn
19:16
15
Rd
15:12
RA1D
RA2D
WA3D
RegWrite
WE3
A1
A2
A3
WD3
R15
RD1
RD2
Reg. File
(PC+8)D
ImmSrc
Extend
11:0
ALUSrc
ALUControl
clk
ForwardAE
ForwardBE
SrcAE
SrcBE
ALU
ALUResultE
WriteDataE
ExtImmE
WA3E
Execute pipeline register
MemWrite
clk
WE
A
RD
Data Memory
WD
ALUOutM
WA3M
Memory pipeline register
clk
MemtoReg
ReadDataW
ALUOutW
WA3W
ResultW
Writeback pipeline register
Fetch
Decode
Execute
Memory
Writeback
ALUResultE
StallF
StallD
FlushD
FlushE
ForwardAE
ForwardBE
Match
RegWriteM
RegWriteE
MemtoRegE
Hazard Unit

Match

Match_1E_M = (RA1E == WA3M)

Match_1E_W = (RA1E == WA3W)

if (Match_1E_M . RegWriteM) → ForwardAE = 10 (AWOutM)

else if (Match_1E_W . RegWriteM) → ForwardAE = 01 (ResultW)

else ForwardAE = 00 ([RA1])

→ Memory stage has priority

* STR and B instructions do not write results to RF (no need to have their results forwarded)
  ↓
  thus use RegWrite W, M signals in the conditions

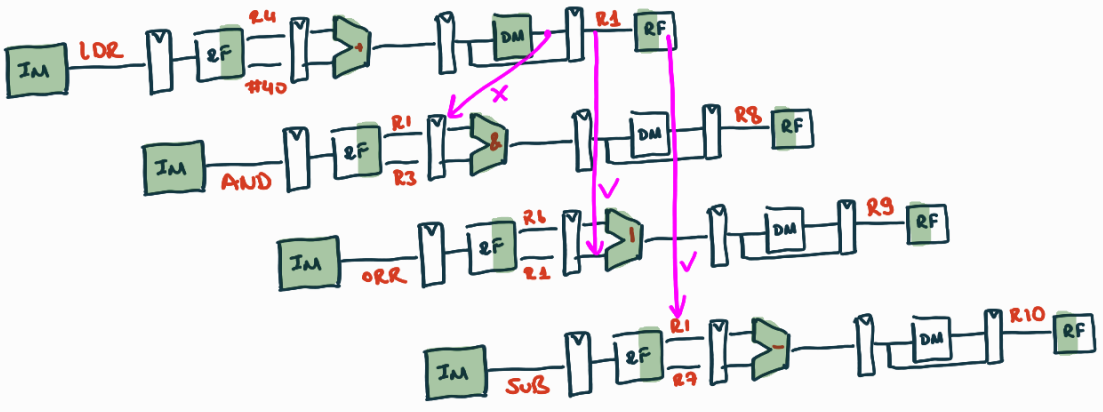
Data Hazard: Example 2

LDR R1, [R4, #40]

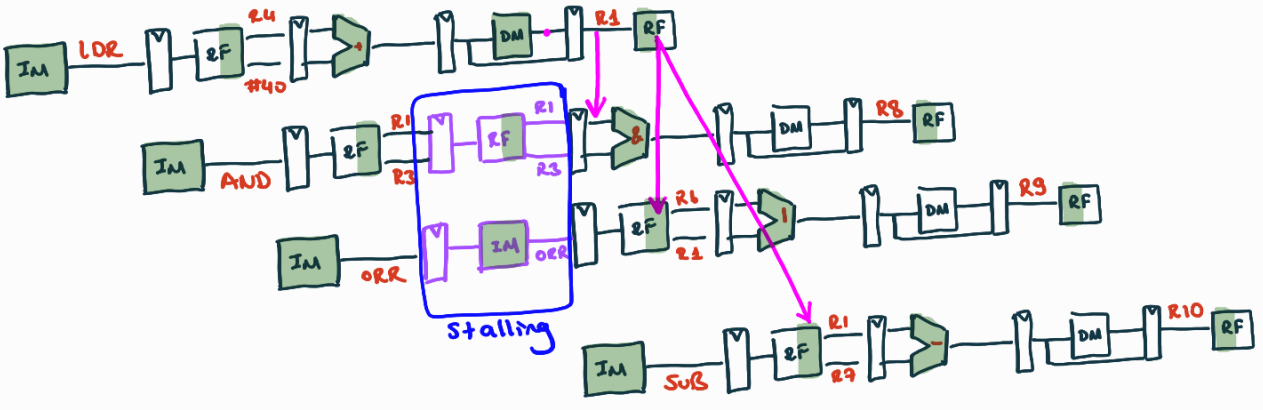
AND R8, R1, R3

ORR R9, R6, R1

SUB R10, R1, R7



Solution: Stalling



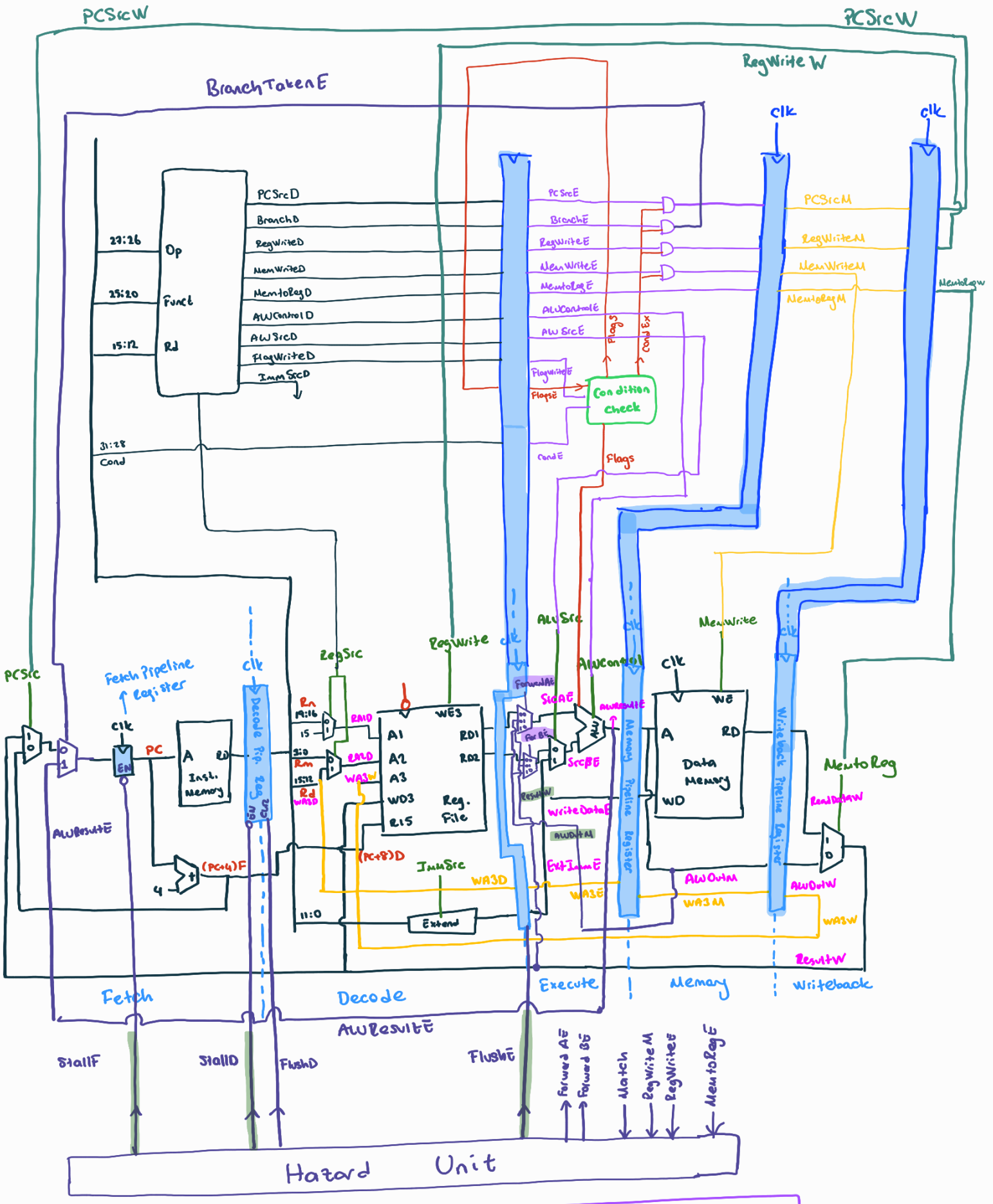
In Decode Stall:

→ Execute stage control signals are zero

→ Execute stage register is flushed

→ All previous stages must be stalled

* The pipeline register directly after the stalled stage must be cleared (flushed)



when LDR occurs

$Match\_12D\_\bar{E} = (RA1D == WA3\bar{E}) + (RA2D == WA3\bar{E})$

$LDRstall = Match\_12D\_\bar{E} \cdot MemtoReg\bar{E}$ → 1 for LDR

$StallF = StallD = FlushE = LDRstall$ → look at the next page

later it will change!

Control Hazard

Branch (B)

→ Instruction after branch fetched before branch occurs

→ These 4 instructions must be flushed if branch happens

↓

! Degrades the system performance

Reducing Branch Misprediction Penalty

Early branch decision: BTA and CondEx are known in Execute stage

Flush 2 instructions

Stall and Flush Branches

→ when branch is taken, flush Decode and Execute

→ When a PC write is in the pipeline → Stall fetch

→ Stall fetch → Flush Decode

→ PCWrPending is asserted when a PC write is in progress

→ During this time, fetch is stalled, decode is flushed

→ when the PCWrite reaches the Writeback stage (PCSrcW asserted), StallF is released to allow the write to occur ✱ but FlushD is still asserted

PCWrPendingF = PCSrcD + PCSrcĒ + PCSrcM → Fetch is stalled, Decode is flushed

StallF = LDRstall + PCWrPendingF

StallD = LDRstall

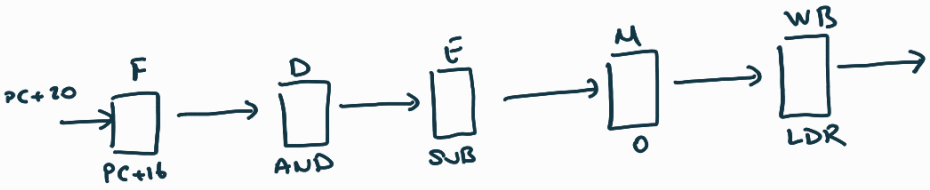
FlushD = PCWrPendingF + PCSrcW + BranchtakenĒ

FlushĒ = LDRStall + BranchtakenĒ ;

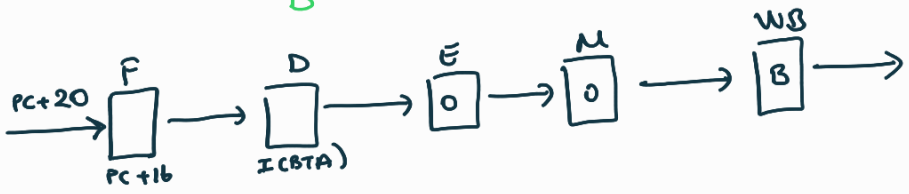
Pipelined Processor Performance

Ideally CPI = 1, however flushes or stalles may occur → waste cycle

LDR CPI = 1 → 2 (for each LDR followed by a Register)



B CPI = 1 → 3



Example:

Suppose

- 25% loads
- 10% stores
- 13% branches
- 52% data processing

- 40% of loads used by next instruction that uses the result
- 50% of branches mispredicted (taken)

→ load CPI = 1 (no stalling) → 2 (when stalling)

CPI lw = 1 x (0.6) + 2 . (0.4) = **1.4** for LDR

→ Branch CPI = 1 (no stalling) → 3 (when stalling)

CPI beq = 1 x 0.5 + 3 x 0.5 = **2**

Avg. CPI = (0.25) x (1.4) + (0.1) 1 + (0.13) x 2 + (0.52) 1 = 1.23
(1.4: LDR, 1: STR, 2: B, 1: DP)

Critical Path

$$T_{cs} = \max \begin{cases} t_{pcq} + t_{mem} + t_{setup} \\ 2\,(t_{REF\,read} + t_{setup}) \\ t_{pcq} + 2t_{mux} + t_{ALU} + t_{setup} \\ t_{pcq} + t_{mem} + t_{setup} \\ 2\,(t_{pcq} + t_{mux} + t_{REF_{setup}}) \end{cases} \begin{matrix} F \\ D \\ E \\ M \\ WB \end{matrix}$$

↓ half cycle work thus multiply with 2