Tahmini Ders İçeriği
(Tentative Couse Schedule – Syllabus)

1. Hafta: Sayı sistemleri, onluk/ikilik taban sayı gösterimleri, mantıksal kapılar, computer system overview, başarım (performance)

2. Hafta: 2'lik tabanda işaretli sayılar, mikroişlemci tarihi, benchmarking, başarım,

3. Hafta: Başarım, Amdahl yasası, RISC-V development Environment, Verilog HDL ile Birleşik (Combinational) devreler

4. Hafta:, Verilog HDL ile sıralı (sequential) mantıksal devre ve sonlu durum makinası tasarımı, timing analysis

5. Hafta: Aritmetik devre tasarımları: Toplama, çıkarma, çarpma, bölme, trigonometri, square-root, hyperbolic, exponential, logarithm

6. Hafta: Fixed ve Floating-Point sayı gösterimleri

7. Hafta: RISC-V buyruk kümesi mimarisi (ISA) ve buyrukların tanıtımı

8. Hafta: RISC-V buyruk kümesi mimarisi (ISA) ve buyrukların tanıtımı

9. Hafta: Vize Çözümleri – CPU Tasarımına Giriş

10. Hafta: Tek-çevrim işlemci tasarımı (single-cycle CPU)

11. Hafta: Çok-çevrim işlemci tasarımı (multi-cycle CPU)

12. Hafta: Boruhatlı işlemci tasarımı (pipelined CPU)

13. Hafta: Bellek sistemi ve hiyerarşisi

14. Hafta: Gömülü sistemler, mikrodenetleyiciler, SoCs

MULTI-CYCLE CPU

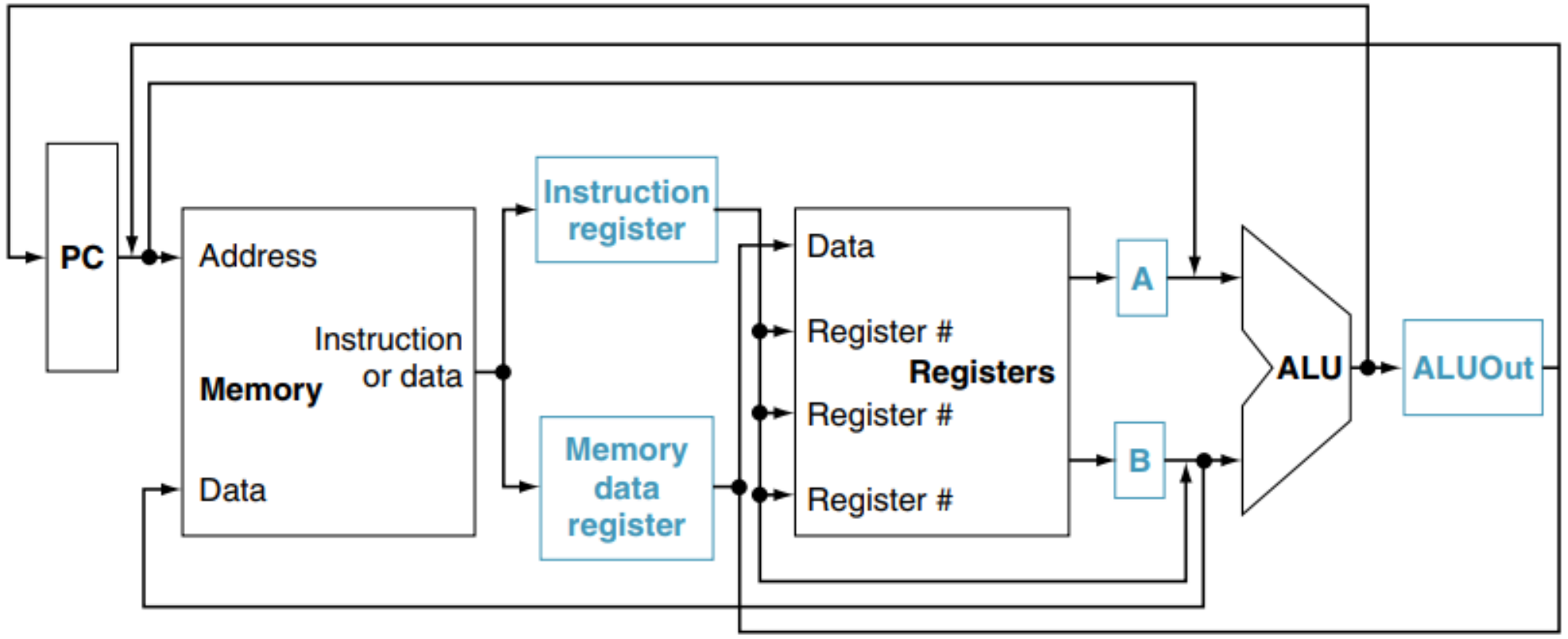


FIGURE e4.5.1 The high-level view of the multicycle datapath. This picture shows the key elements of the datapath: a shared memory unit, a single ALU shared among instructions, and the connections among these shared units. The use of shared functional units requires the addition or widening of multiplexors as well as new temporary registers that hold data between clock cycles of the same instruction. The additional registers are the Instruction register (IR), the Memory data register (MDR), A, B, and ALUOut.

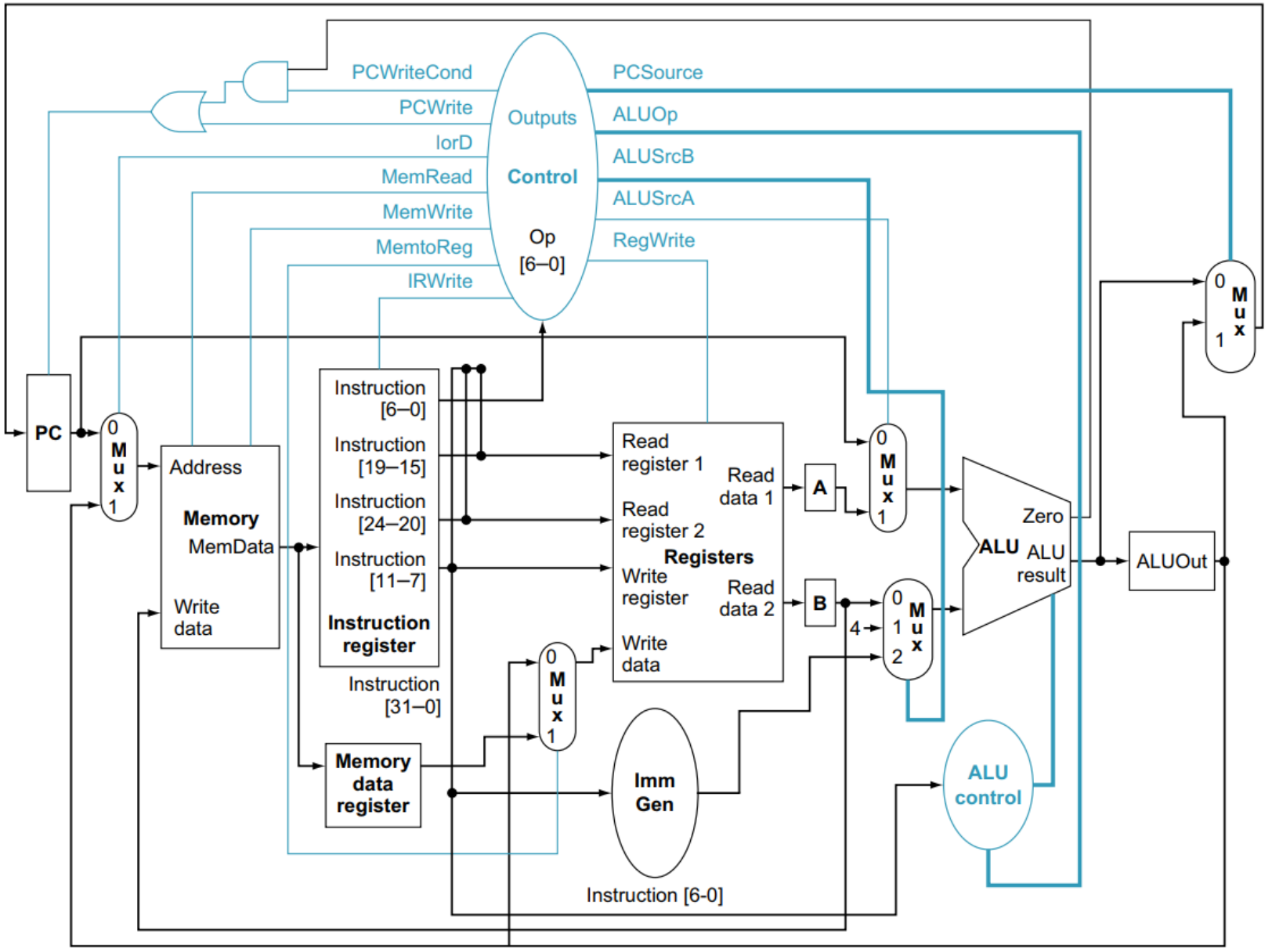


FIGURE e4.5.4 The complete datapath for the multicycle implementation together with the necessary control lines. The control lines of Figure e4.5.3 are attached to the control unit, and the control and datapath elements needed to effect changes to the PC are included. The major additions from Figure 4.29 include the multiplexor used to select the source of a new PC value; gates used to combine the PC write signals; and the control signals PCSource, PCWrite, and PCWriteCond. The PCWriteCond signal is used to decide whether a conditional branch should be taken.

MULTI-CYCLE CPU – EXECUTION STEPS

1. Adım - Instruction Fetch:

Fetch instruction from memory

2. Adım – Instruction Decode and Register Fetch:

Read registers and decode the instruction

3. Adım - Execution, memory address computation, or branch completion

Execute the operation or calculate an address

4. Adım - Memory access or R-type instruction completion step

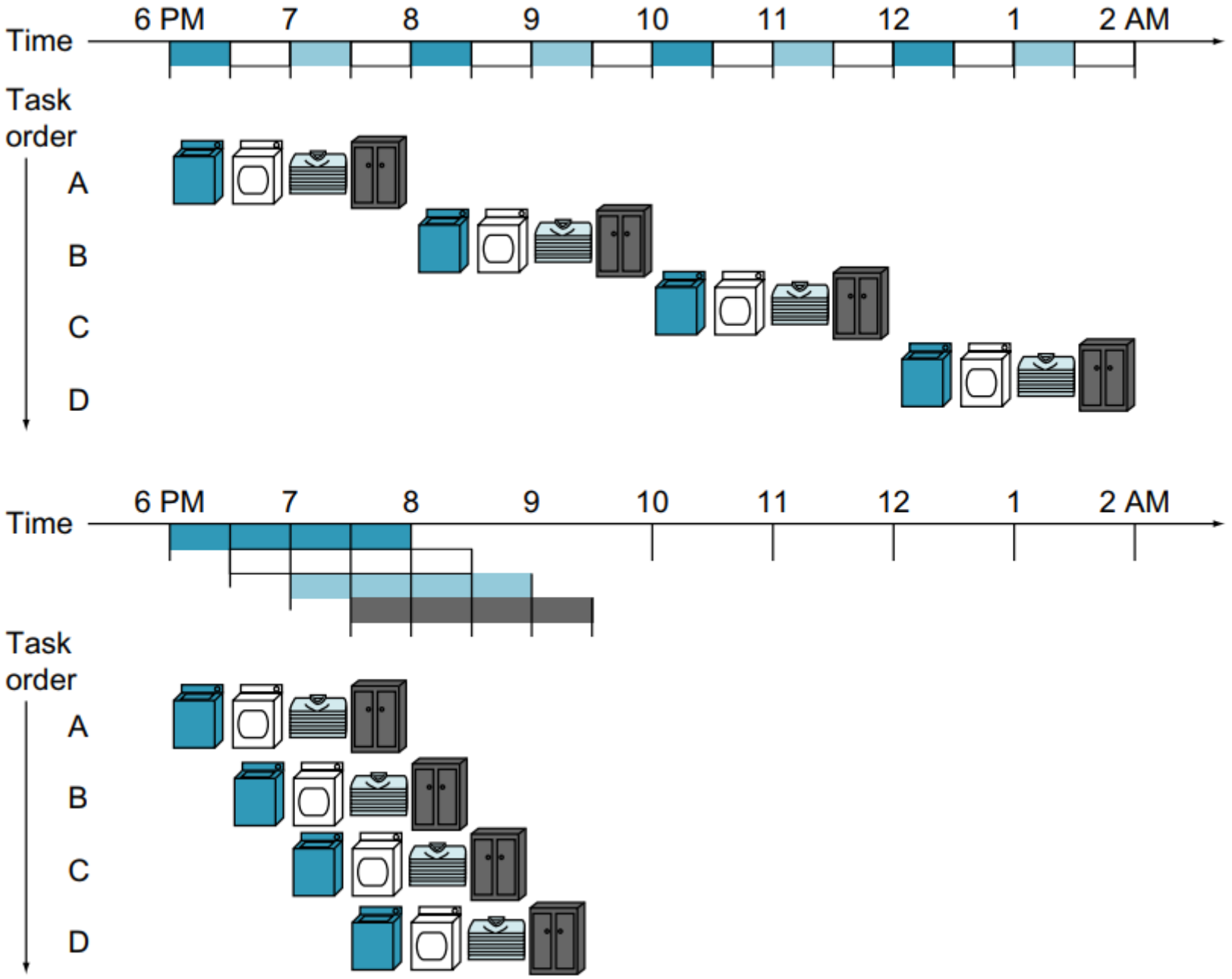
Access an operand in data memory (if necessary)

5. Adım - Memory read completion step

Write the result into a register (if necessary)

Fetch – Decode – Execute – Memory – Writeback

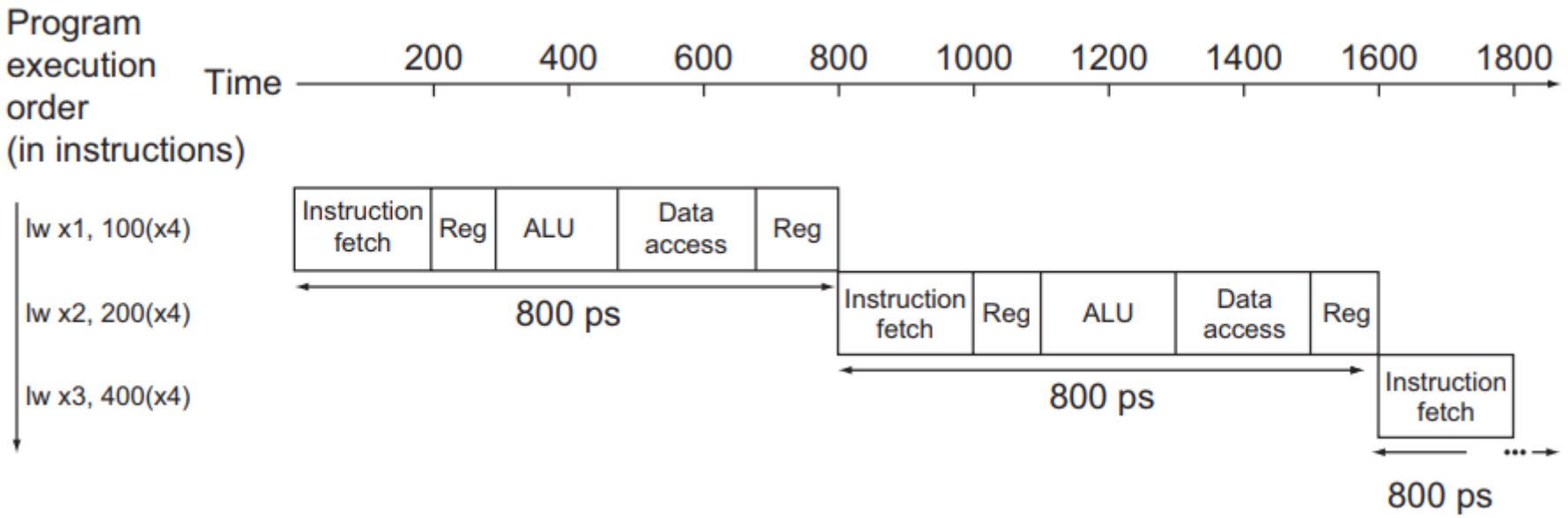
PIPELINING EXAMPLE



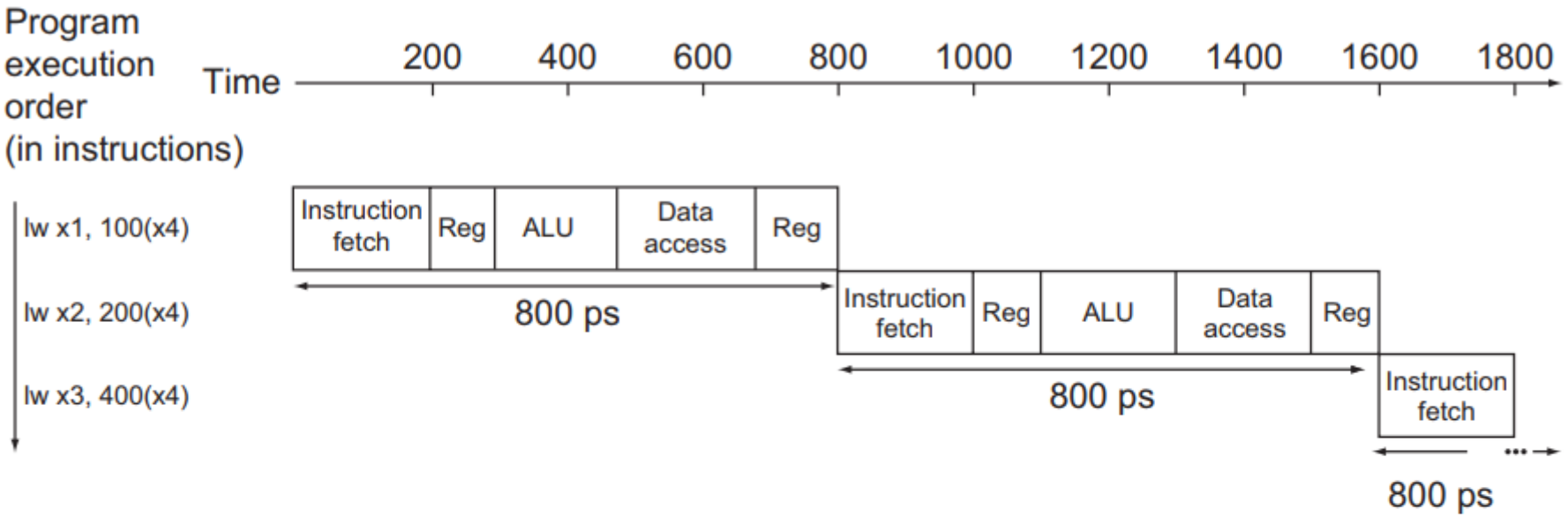
PIPELINING EXAMPLE

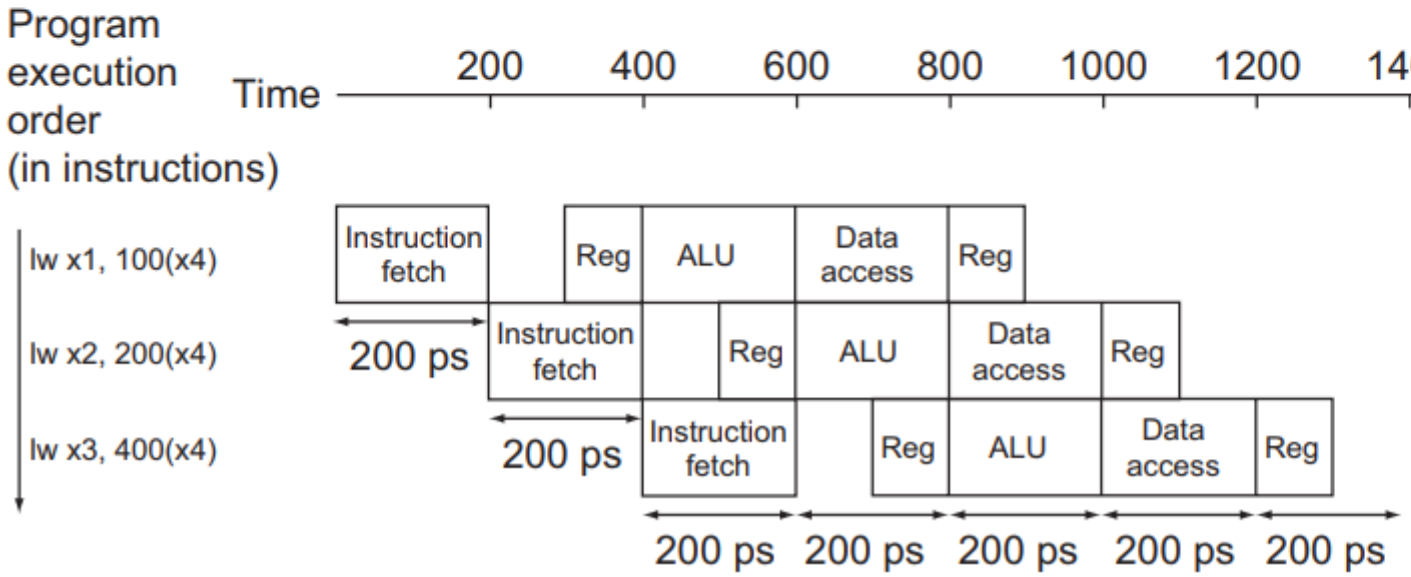
SINGLE-CYCLE vs PIPELINING

Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (lw)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (sw)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (add, sub, and, or)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (beq)	200 ps	100 ps	200 ps			500 ps



SINGLE-CYCLE vs PIPELINING





PIPELINING – BAZI NOTLAR

1) Pipelining ile 1 instruction'un tamamlanma süresi (latency) azalmaz, sistemin toplam performansı (throughput) artar

2) Teorik olarak en iyi durumda, sistemin performansı, pipeline adım sayısı katı kadar artar. Örneğin 1 ms süren bir program 5 aşamalı bir pipeline yapılırsa en iyi durumda 200 us, 10 aşamalı bir pipeline yapılırsa en iyi durumda 100 us sürer, yani 10 kat hızlanır. Bunun için saat periyodunun eşit bölünebilmesi gerekir.

$$\text{CPU time} = \text{Instruction count} \times \text{CPI} \times \text{Clock cycle time}$$

$$\frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution time}_Y}{\text{Execution time}_X} = n$$

3) Pratikte bu hızlanma neredeyse imkansızdır. Çünkü instruction'ın veriyolunda geçtiği aşamaları eşit süreli adımlara bölmek mümkün değildir. Bu durumda en yavaş adıma göre saat frekansı belirlenir, hızlanma da bu oranda olur.

4) Teorik olarak pipeline olan bir işlemcide her adımda, yani her saat vuruşunda bir instruction tamamlanır (commit). Yani CPI 1 olur. Pratikte bu imkansızdır. Çünkü branch instruction tamamlanmadan sonraki instruction getirilemez (fetch), ya da yürütülecek olan instruction, bir önceki instruction'un register file'a yazacağı veriyi kullanacaksa önceki instruction'un writeback aşamasının bitmesini beklemek durumunda kalacaktır.

PIPELINE HAZARDS

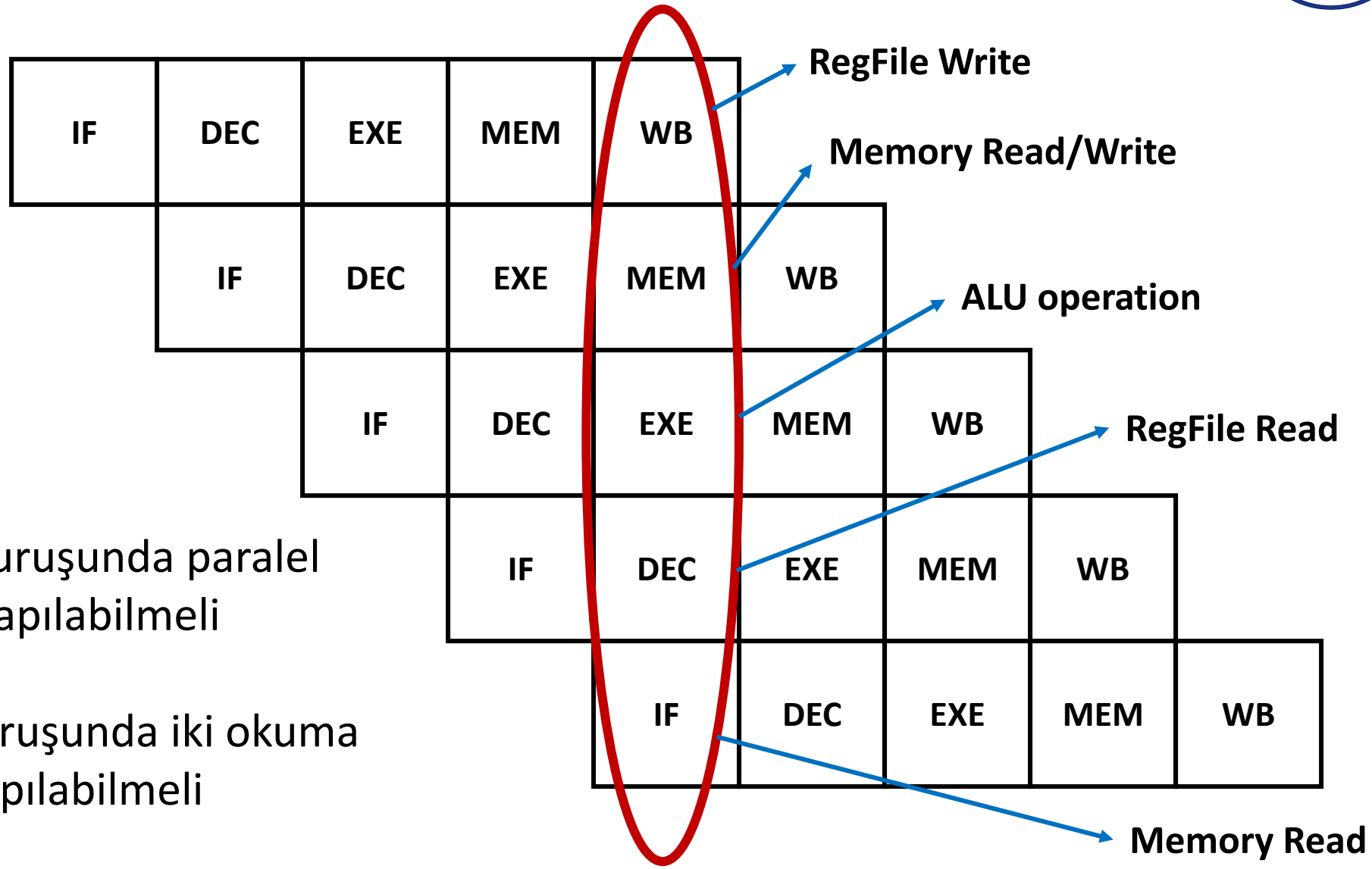
Pipeline ile ideal olan CPI'nın 1 olmasıdır. Fakat bazı pipeline tehlikeleri yüzünden bu mümkün olmamaktadır.

1) Structural Hazards: When a planned instruction cannot execute in the proper clock cycle because the hardware does not support the combination of instructions that are set to execute.

2) Data Hazards: When a planned instruction cannot execute in the proper clock cycle because data that are needed to execute the instruction are not yet available.

3) Control Hazards: When the proper instruction cannot execute in the proper pipeline clock cycle because the instruction that was fetched is not the one that is needed; that is, the flow of instruction addresses is not what the pipeline expected.

STRUCTURAL HAZARDS



Register File'a aynı saat vuruşunda paralel olarak yazma ve okuma yapılabilmeli

Memory'den aynı saat vuruşunda iki okuma ya da okuma ve yazma yapılabilmeli

DATA HAZARDS

Pipeline'daki bir instruction, daha önceden işlenmeye başlayan ama hala pipeline'da olan başka bir instruction'a bağımlı ise veri tehlikesi (data hazard) meydana gelir.

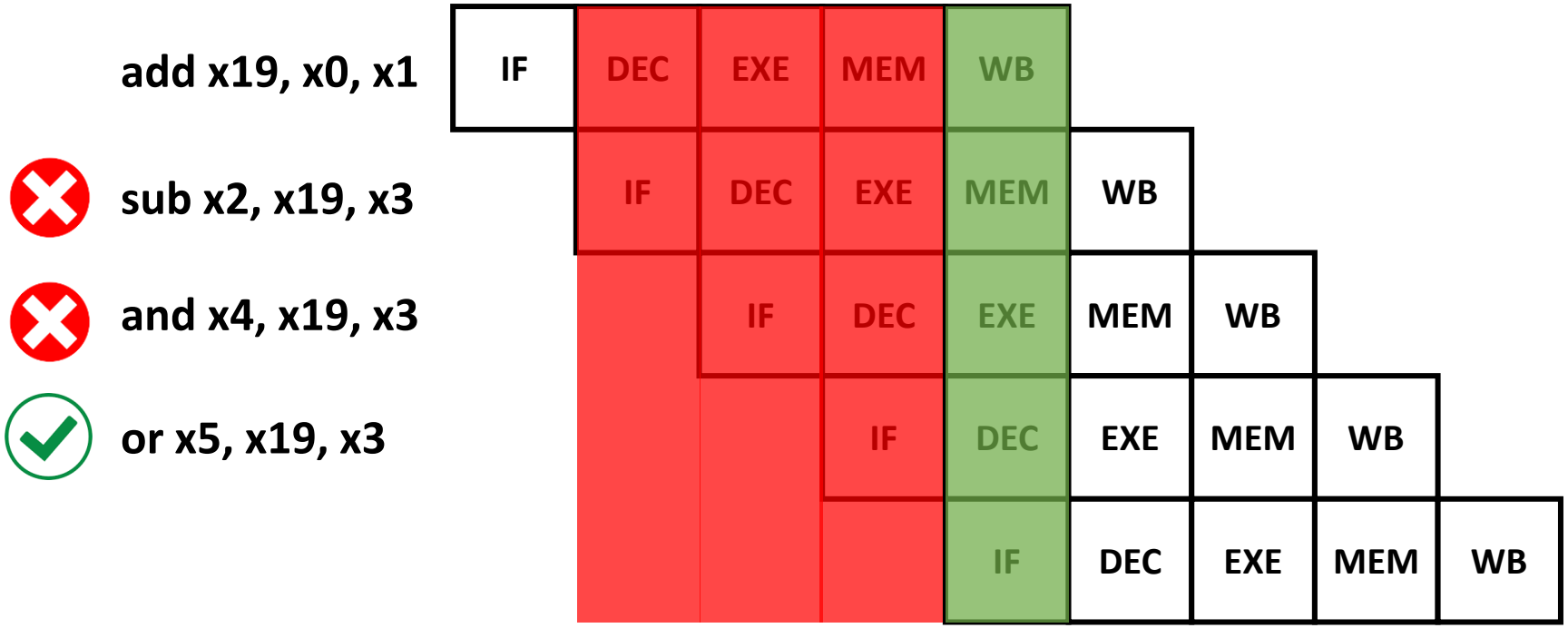
Örnek:

add x19, x0, x1

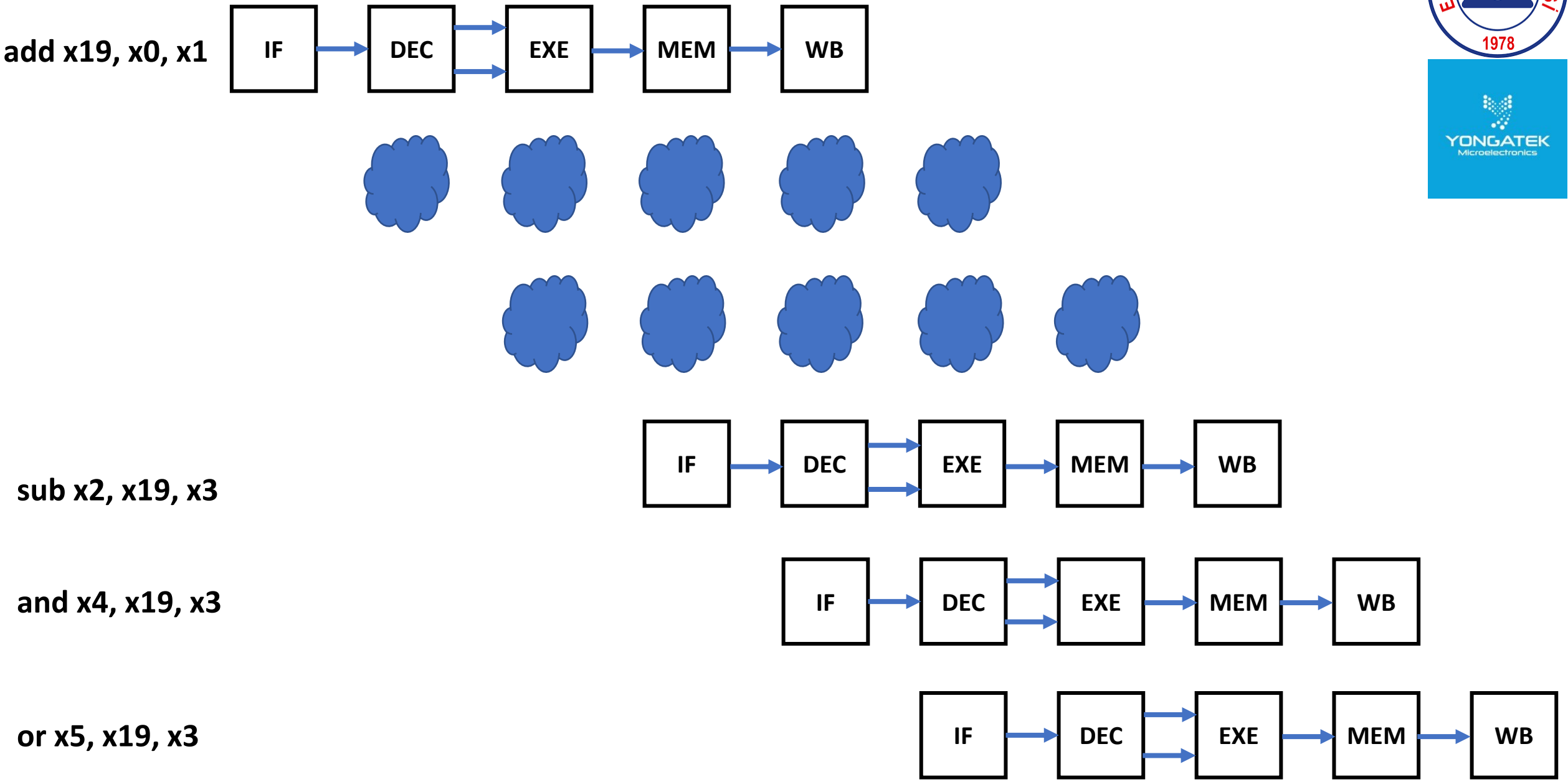
sub x2, x19, x3

and x4, x19, x3

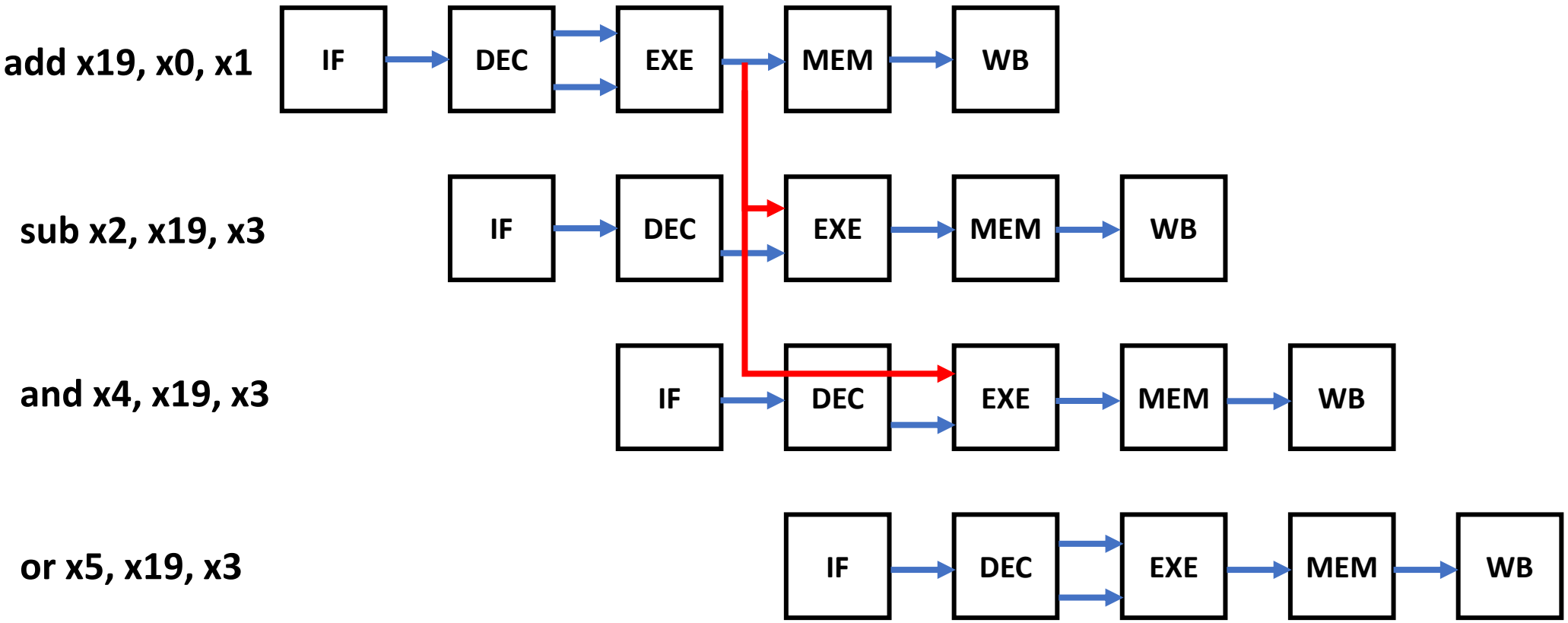
or x5, x19, x3



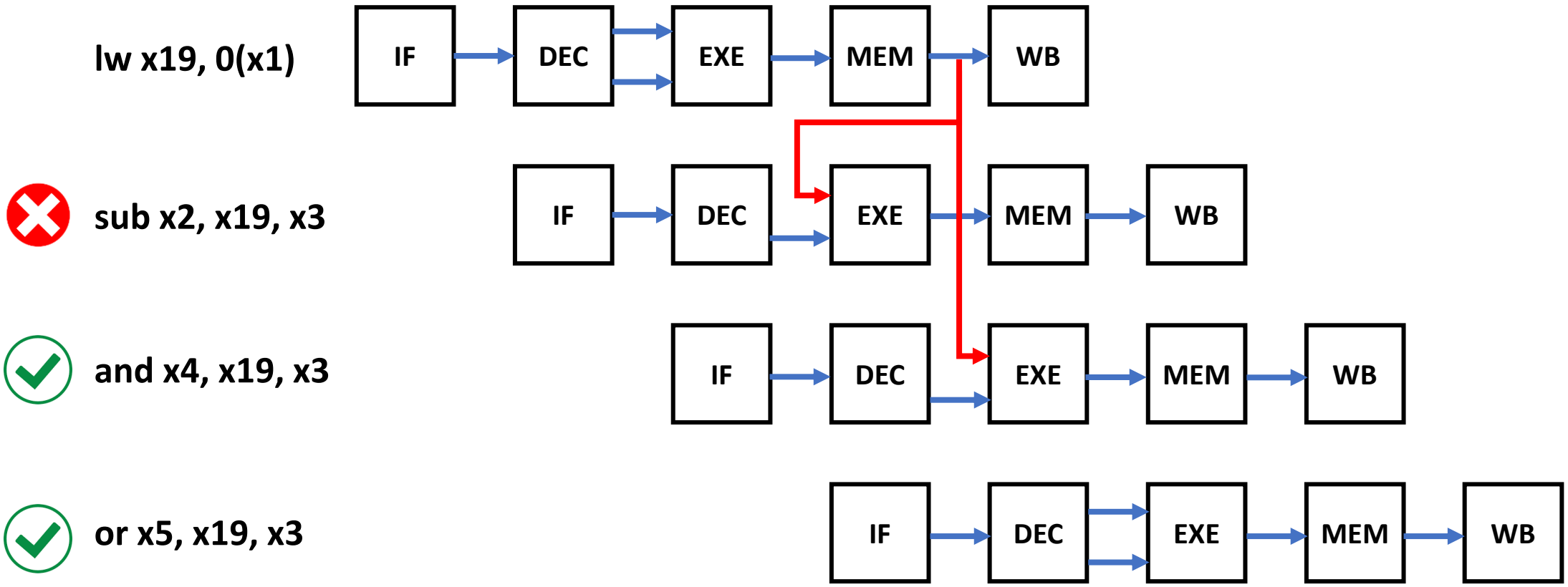
DATA HAZARDS – BUBBLE/STALL



DATA HAZARDS - FORWARDING



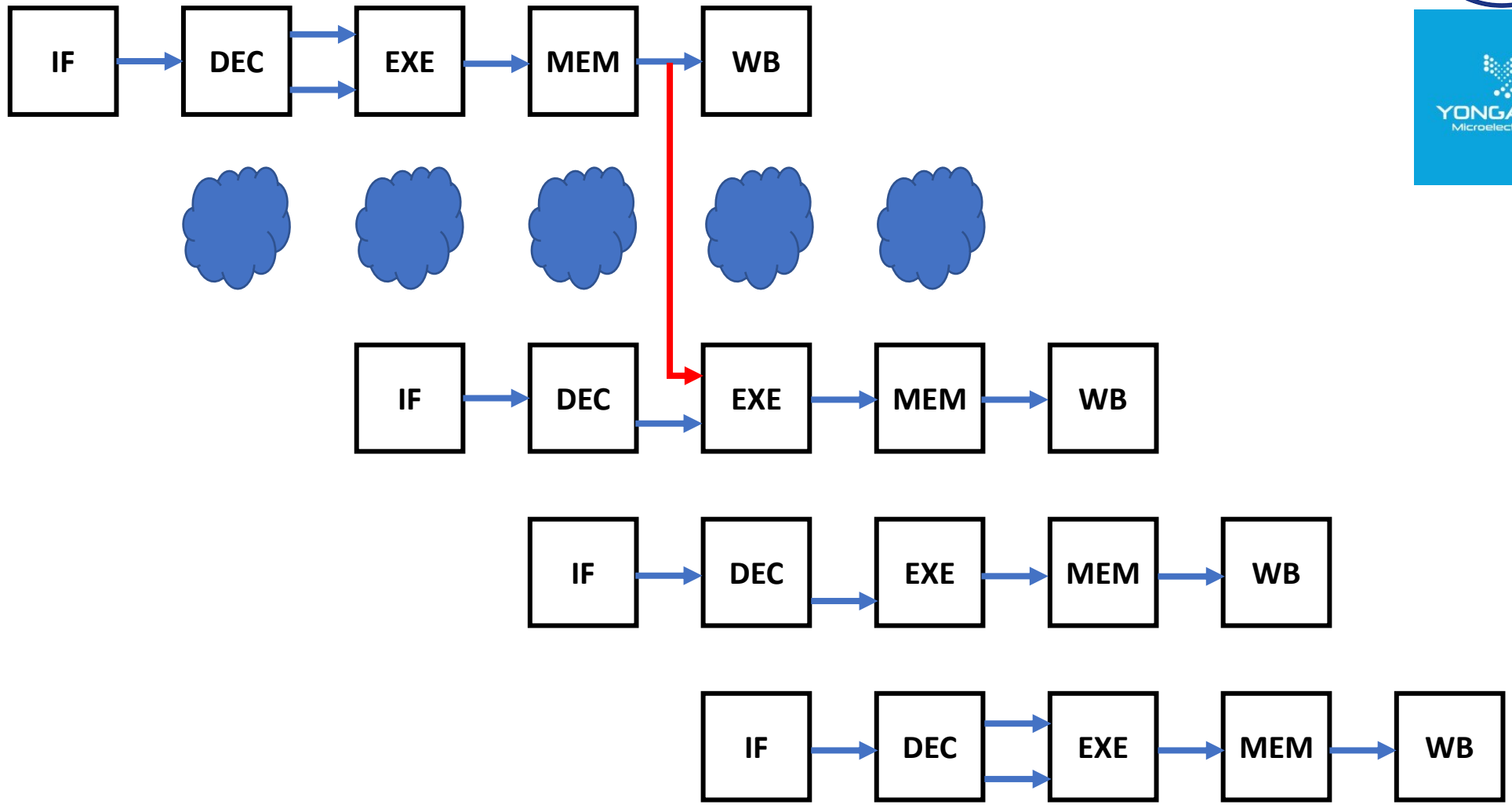
LOAD DATA HAZARDS – FORWARDING ??



YONGATEK Microelectronics

ERCİYES ÜNİVERSİTESİ 1978

LOAD DATA HAZARDS – FORWARDING ??



DATA HAZARDS – COMPILER EFFECT

Reordering Code to Avoid Pipeline Stalls

Consider the following code segment in C:

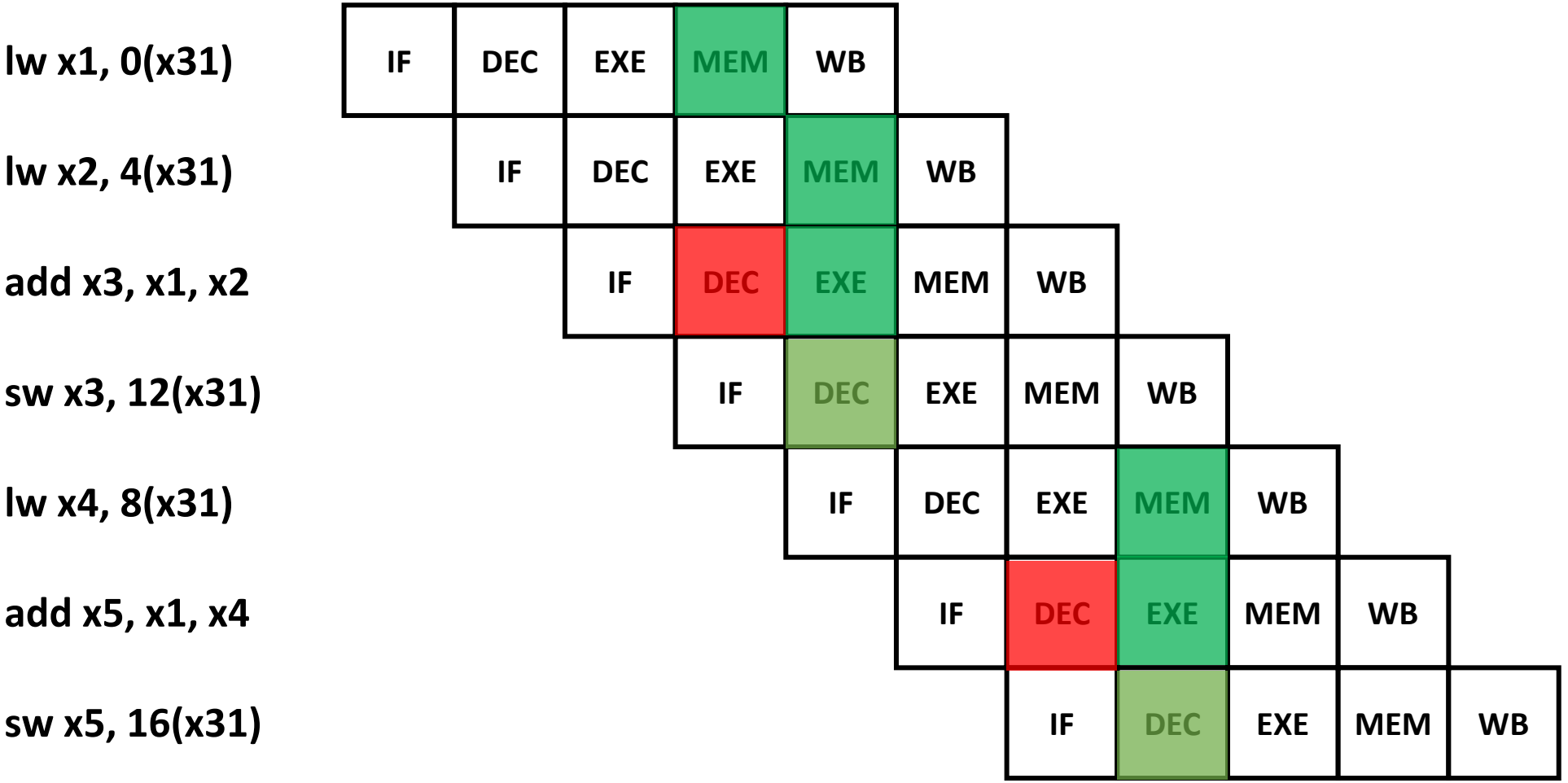
```
a = b + e;
c = b + f;
```

Here is the generated RISC-V code for this segment, assuming all variables are in memory and are addressable as offsets from `x31`:

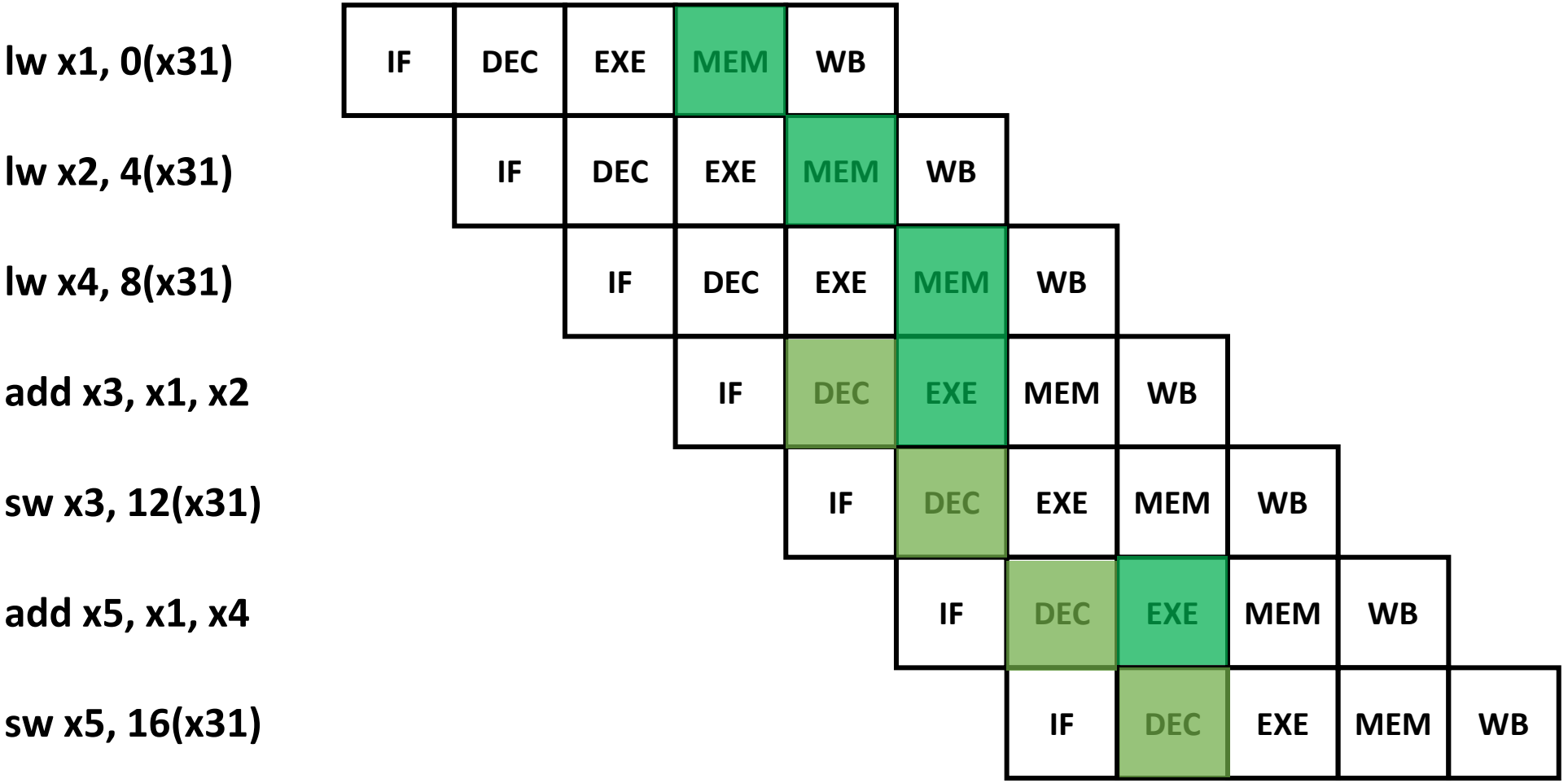
```
lw      x1, 0(x31)   // Load b
lw      x2, 8(x31)   // Load e
add     x3, x1, x2   // b + e
sw      x3, 24(x31)  // Store a
ld      x4, 16(x31)  // Load f
add     x5, x1, x4   // b + f
sw      x5, 32(x31)  // Store c
```

Find the hazards in the preceding code segment and reorder the instructions to avoid any pipeline stalls.

DATA HAZARDS – COMPILER EFFECT



DATA HAZARDS – COMPILER EFFECT



CONTROL HAZARDS

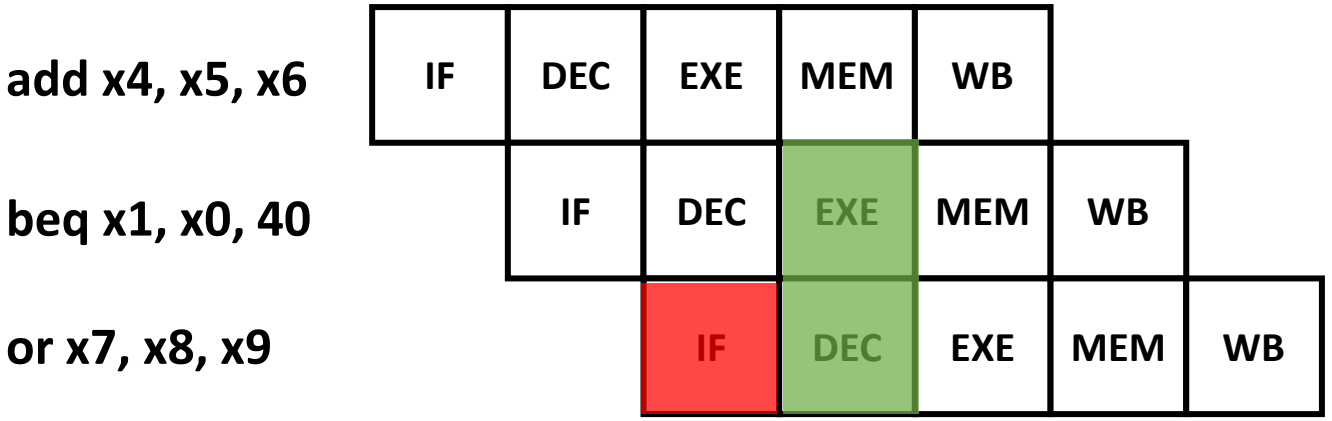
Pipeline’daki bir instructionın sonucuna göre bir sonraki saat vuruşunda hangi instructionun fetch edileceği durumu belirsizliği varsa kontrol tehlikesi mevcuttur.

Örnek:

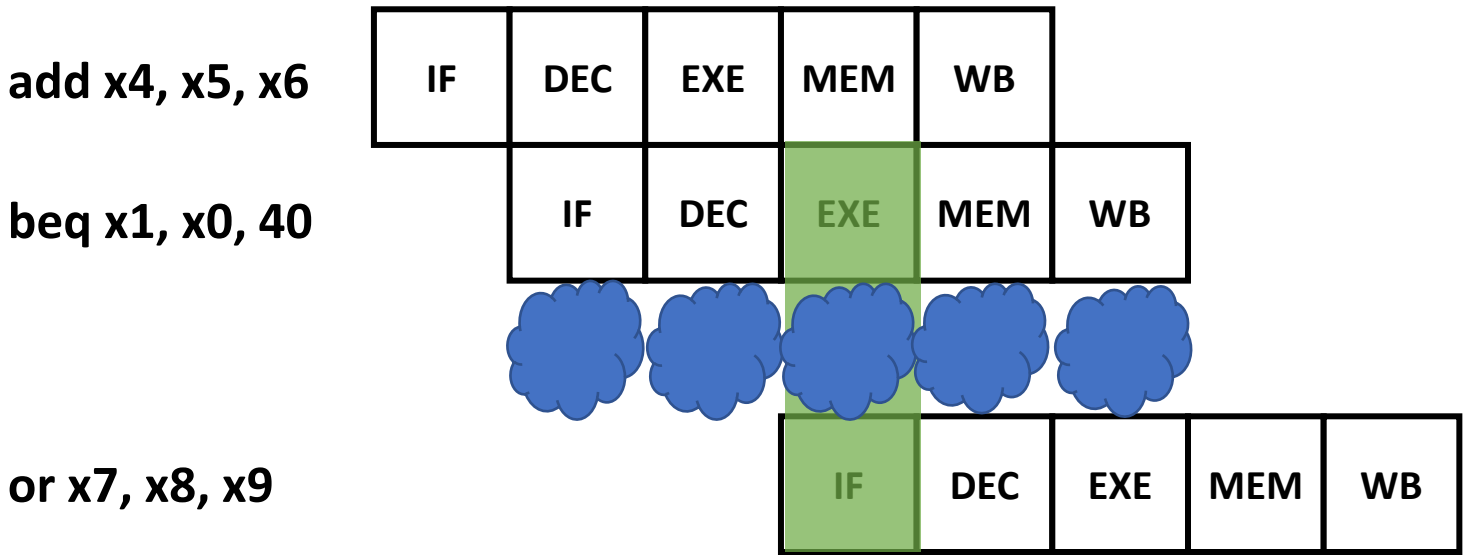
add x4, x5, x6

beq x1, x0, 40

or x7, x8, x9



CONTROL HAZARDS - BUBBLE



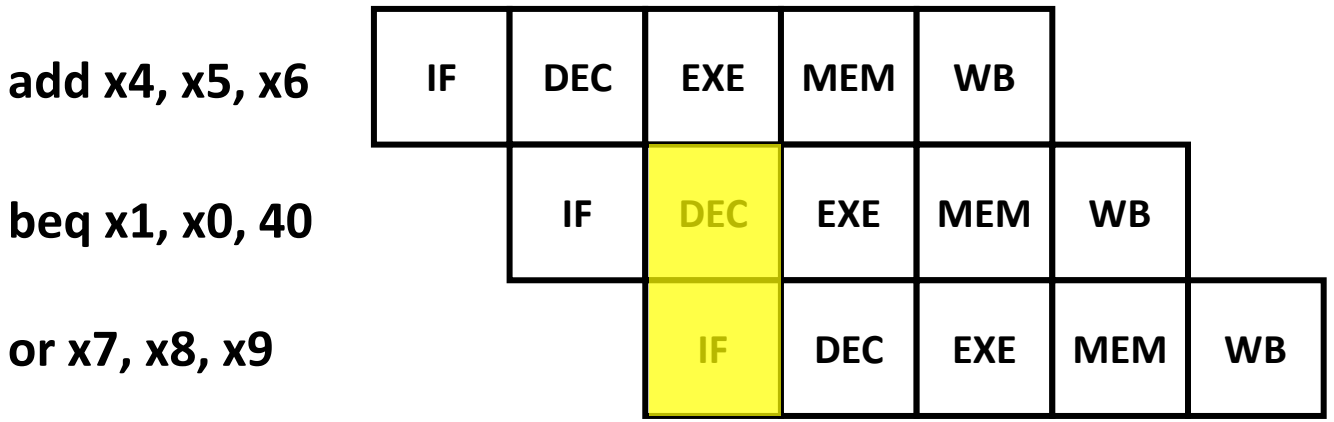
CONTROL HAZARDS - PREDICT

Branch instruction'ının doğru olduğunu kabul ederek bir sonraki instructionı ilgili memory adresinden getiriyoruz

Eğer doğru tahminde bulunduysak sorun yok, işlemci hiç beklemeden işleyişe devam etmiş oldu.

Peki ya yanlış tahminde bulunulduysa, yani x0 ve x1 eşit değilse ?

Bu durumda pipeline boşaltırılır (flush) ve branch prediction sonrası yanlış getirilen instruction için yapılan işlemler iptal edilir. Doğru instruction fetch edilip işleme devam edilir.



BRANCH PREDICTION METHODS

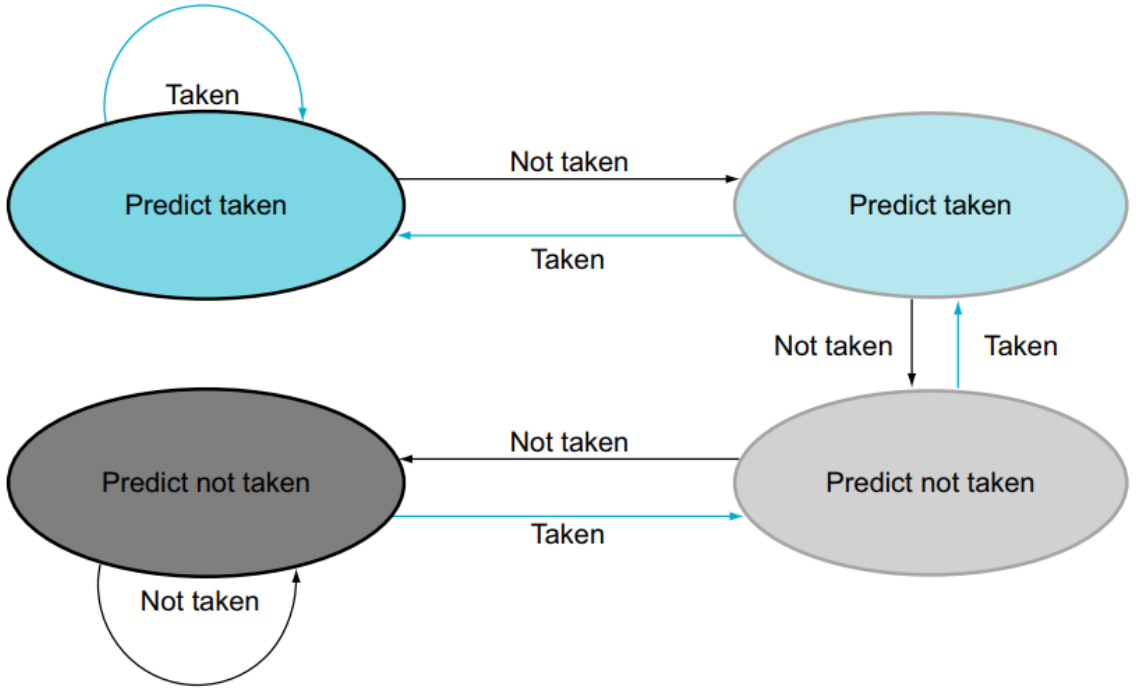
Branch prediction yöntemleri temelde 2'ye ayrılır: Static ve dynamic

Static branch prediction yönteminde branch taken ya da not taken olarak baştan kabul edilir ve bu şekilde branch instruction'dan sonra hangi instruction getirilecekse ona göre hareket edilir. Genel olarak branch not taken almak daha isabetli olur (bir for döngüsünü düşünün)

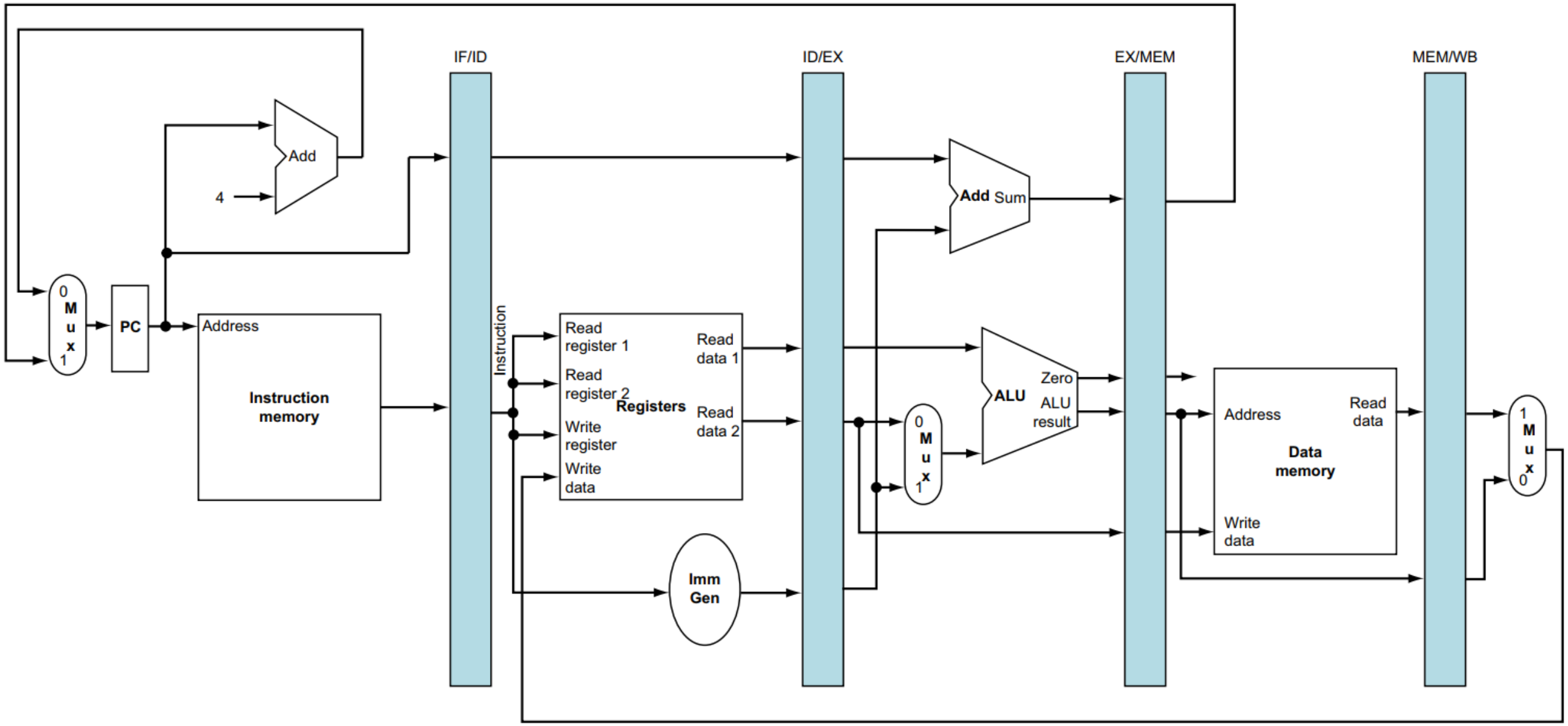
Dinamic branch prediction'da ise yapılan tahminlerin başarı oranı bir bellekte tutulur ve geçmiş tahminlerin isabet oranına göre bir sonraki branch tahmini değişkenlik gösterebilir. Örnek olarak son 3 branch instruction'ı taken olarak tahmin edilip hatalı çıktıysa 4. tahmini de taken mı alırsınız yoksa not taken mı?

Branch tahmin algoritmaları işlemci performansında çok önemli bir rol oynamaktadır. Bu konuda çok detaylı araştırmalar yapılmıştır ve yapılmaya devam etmektedir.

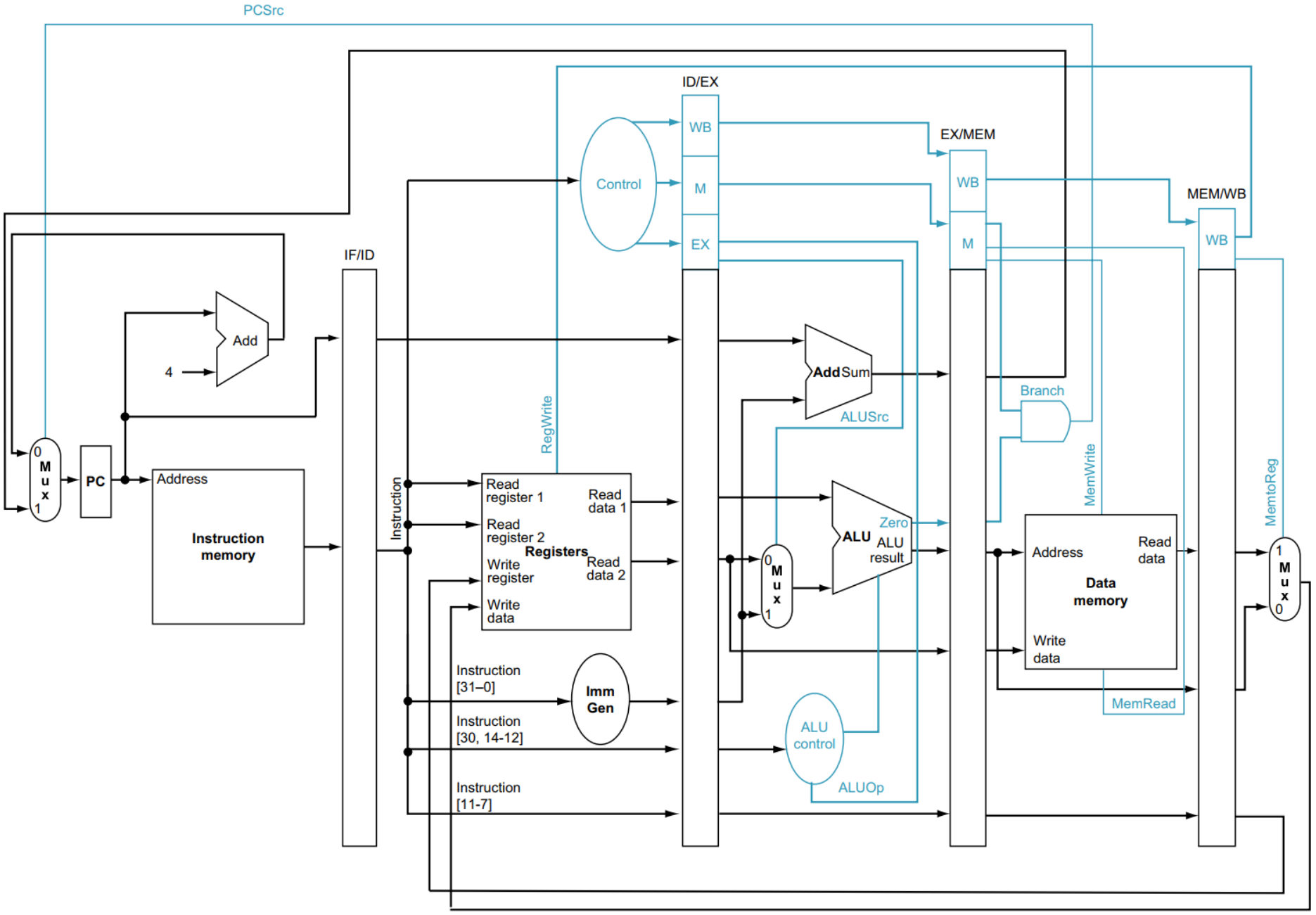
2-bit ile geçmiş branch tahmin ve sonuçlarını tutan bir branch prediction state machine örneği:



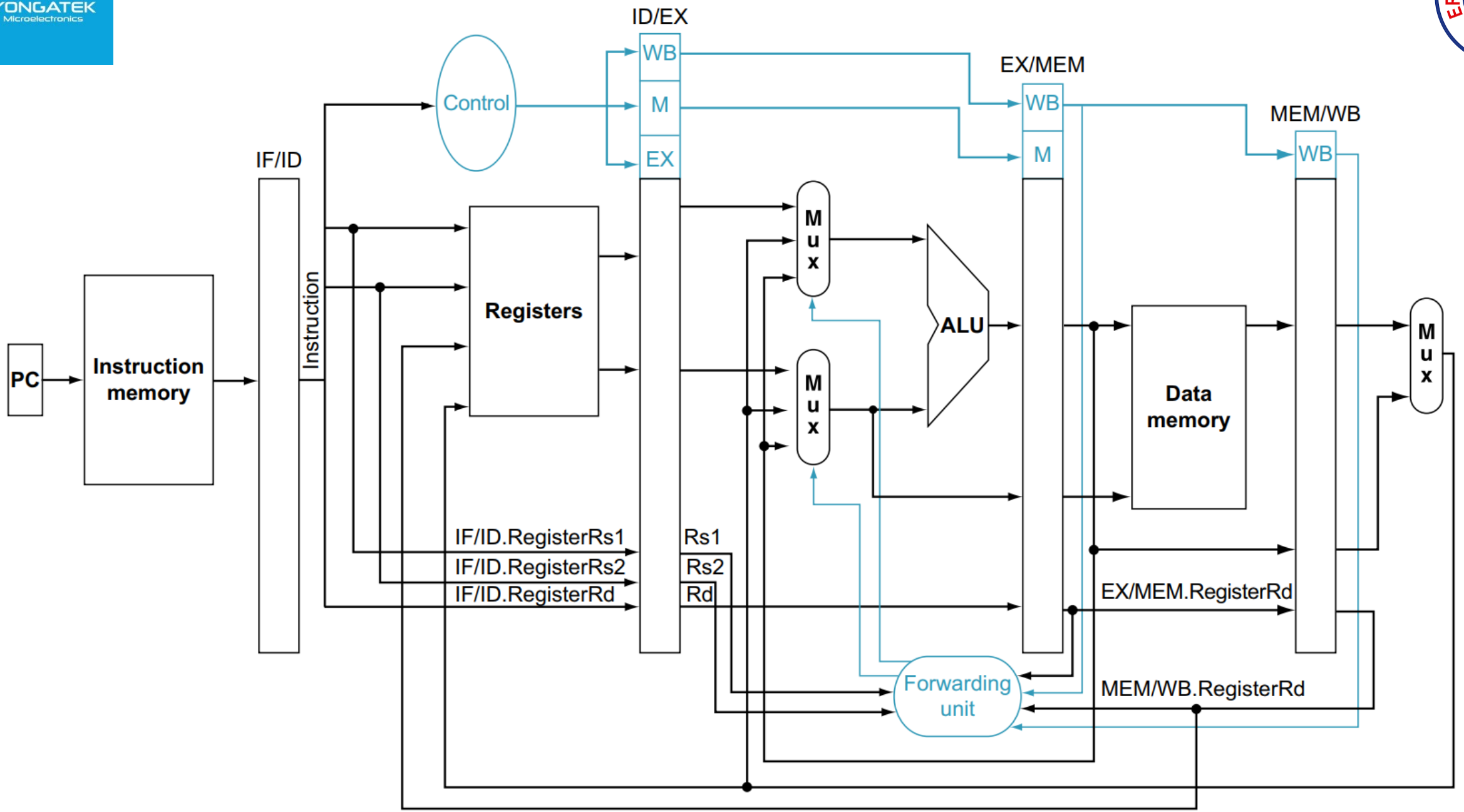
PIPELINED DATAPATH



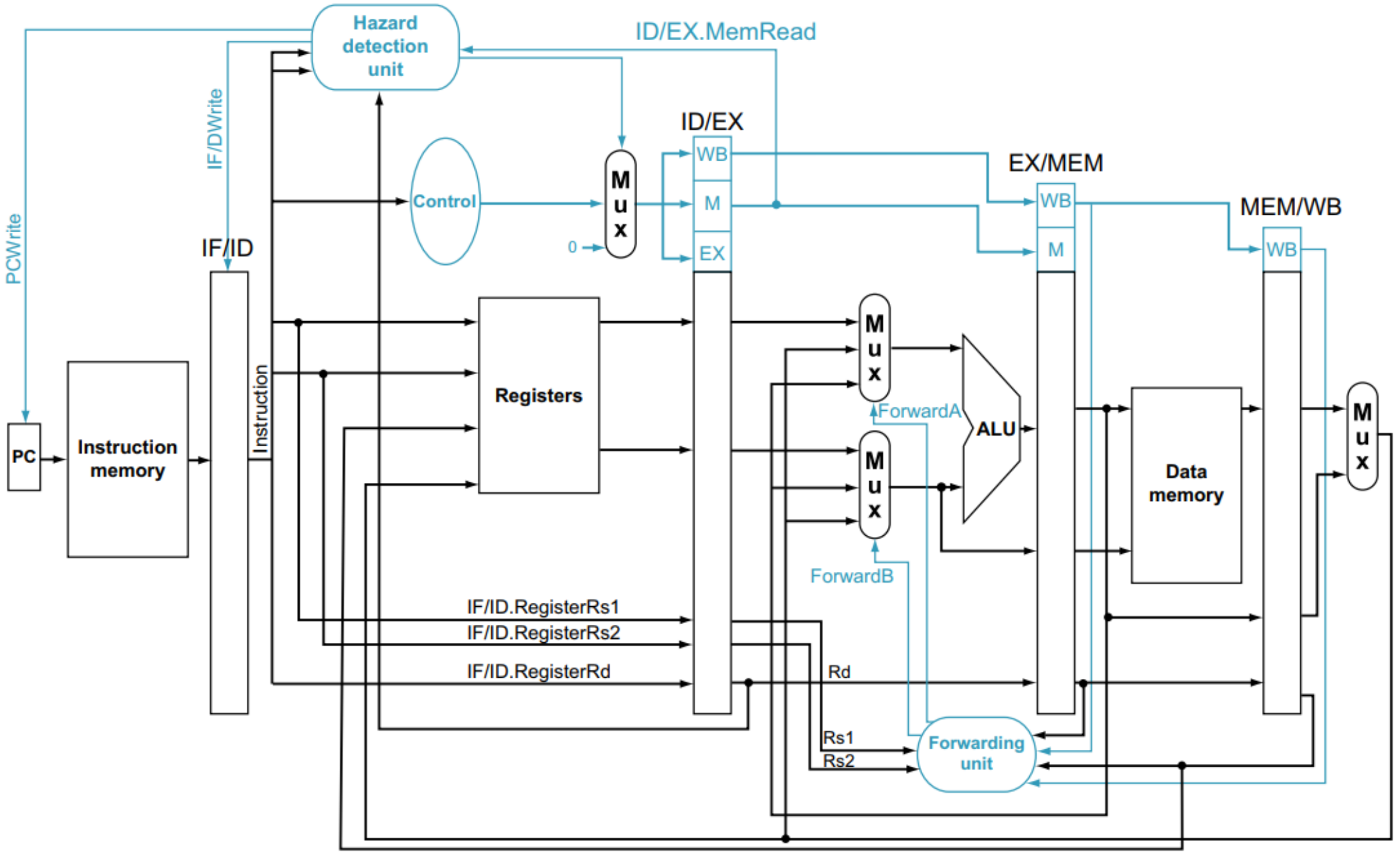
PIPELINED CONTROL



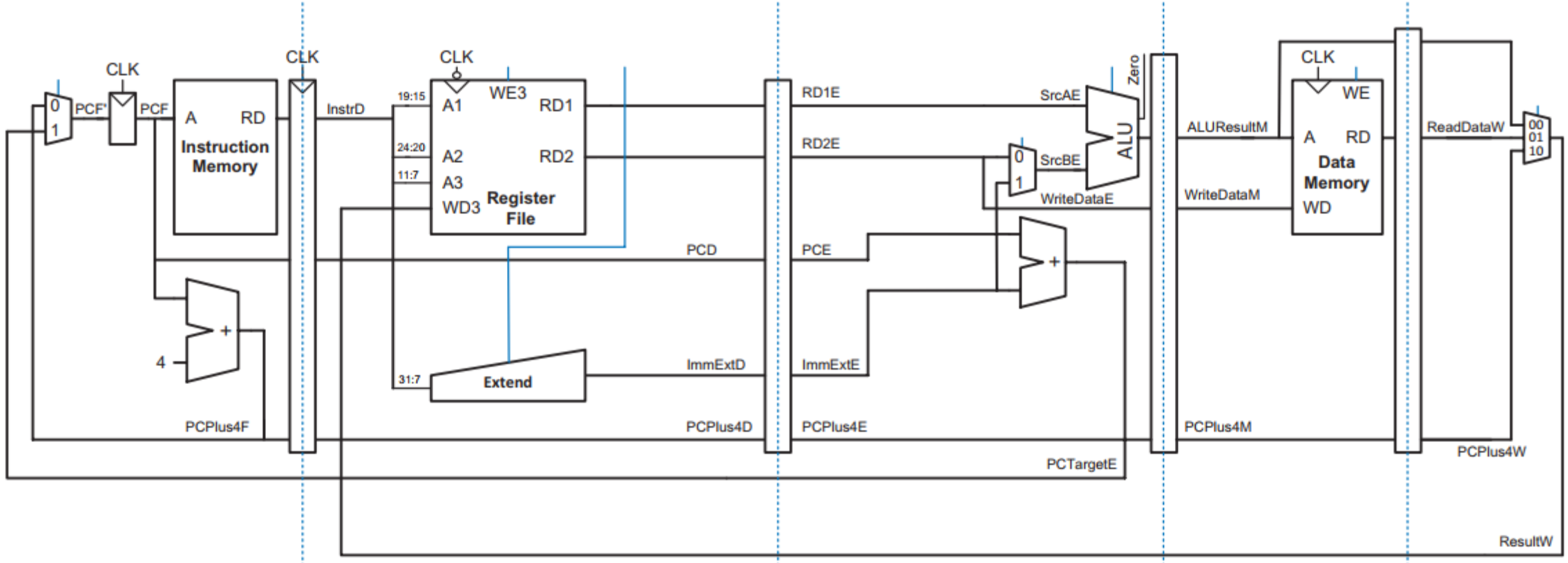
FORWARDING UNIT



HAZARD DETECTION UNIT



A PIPELINE PROBLEM !



A PIPELINE PROBLEM !