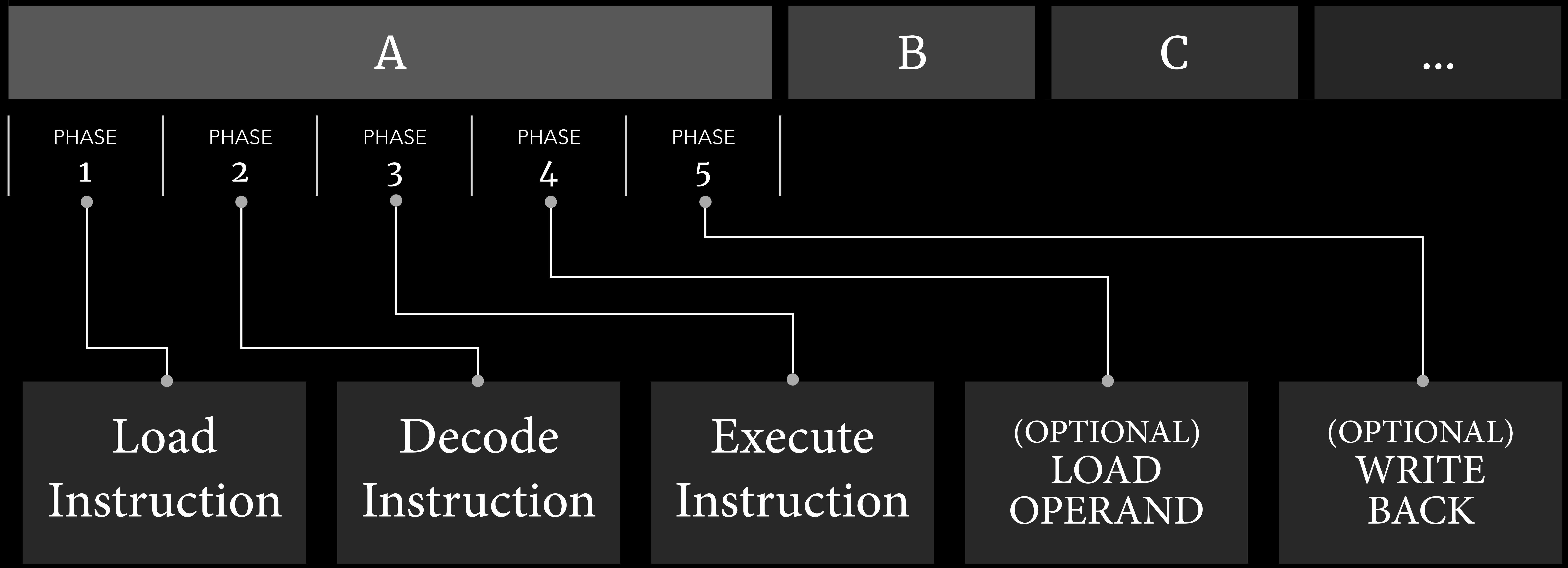
X86 Instruction Pipeline Mechanism

张宇轩, 廖开新

Nov 13, 2019

Why Introducing Pipeline



Typical Ins-by-Ins Workflow

Why Introducing Pipeline

Pipeline	CLK 1	CLK 2	CLK 3	CLK 4	CLK 5	CLK 6
Load Ins.	A	B	C	D	E	F
Decode Ins.	…	A	B	C	D	E
Execute Ins.		…	A	B	C	D
(OPTIONAL) LOAD OPERAND			…	A	B	C
(OPTIONAL) WRITE BACK				…	A	B

Interlaced Processing (Pipelining)

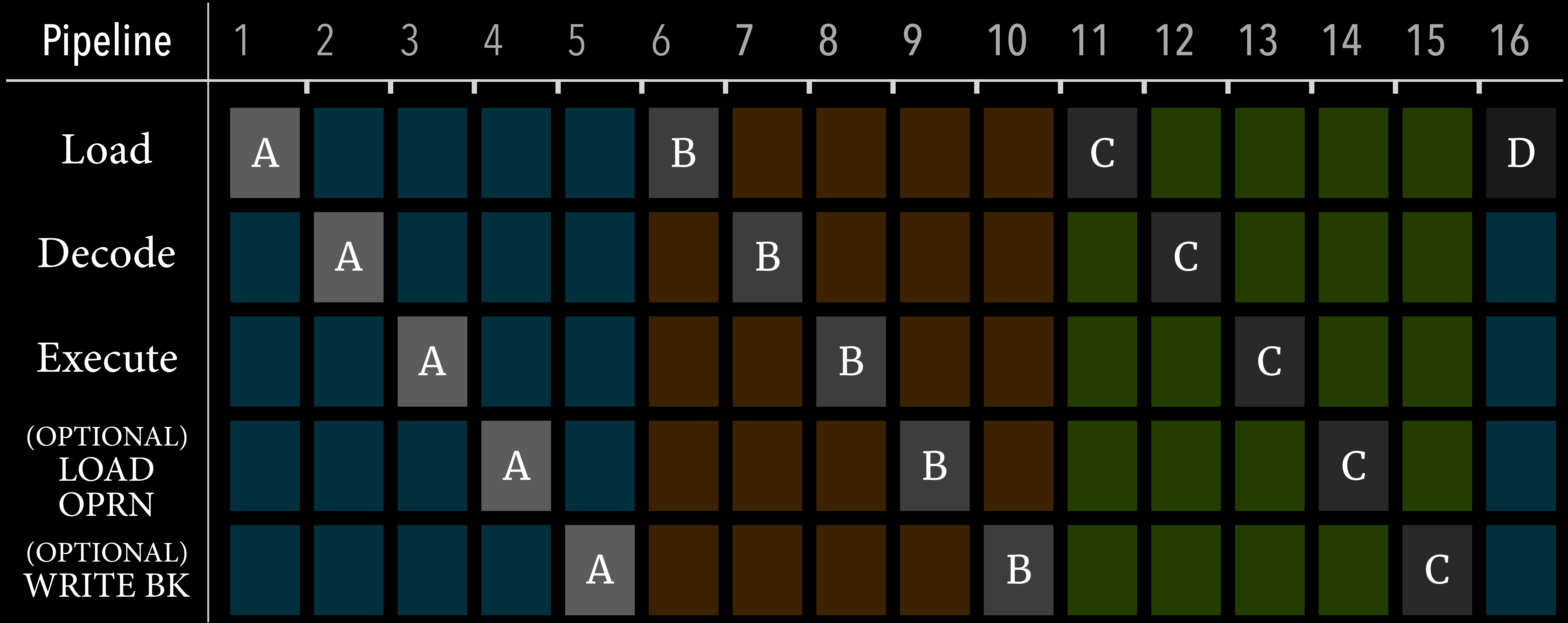
Pipeline Efficiency

Assuming:

- Any instructions can be divided similarly. (N)
- Each stage consume same CPU CLK periods.
- There is no conditional branches which may brake the pipeline. (JE, JS, etc.)

Pipeline	1	2	3	4	5	6
Load	A	B	C	D	E	B’
Decode		A	B	C	D	E
Execute			A	B	C	D
(OPTIONAL) LOAD OPRN				A	B	C
(OPTIONAL) WRITE BK					A	B

Pipeline Efficiency



Pipeline	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Load	A					B					C					D
Decode		A					B					C				
Execute			A					B					C			
(OPTIONAL) LOAD OPRN				A					B					C		
(OPTIONAL) WRITE BK					A					B					C	

Typical workflow viewed in sequential order

Pipeline Efficiency

Pipeline	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Load	A	B	C	D	…											
Decode	…	A	B	C	D	…										
Execute		…	A	B	C	D	…									
(OPTIONAL) LOAD OPRN			…	A	B	C	D	…								
(OPTIONAL) WRITE BK				…	A	B	C	D	…							

In comparison, the improved workflow

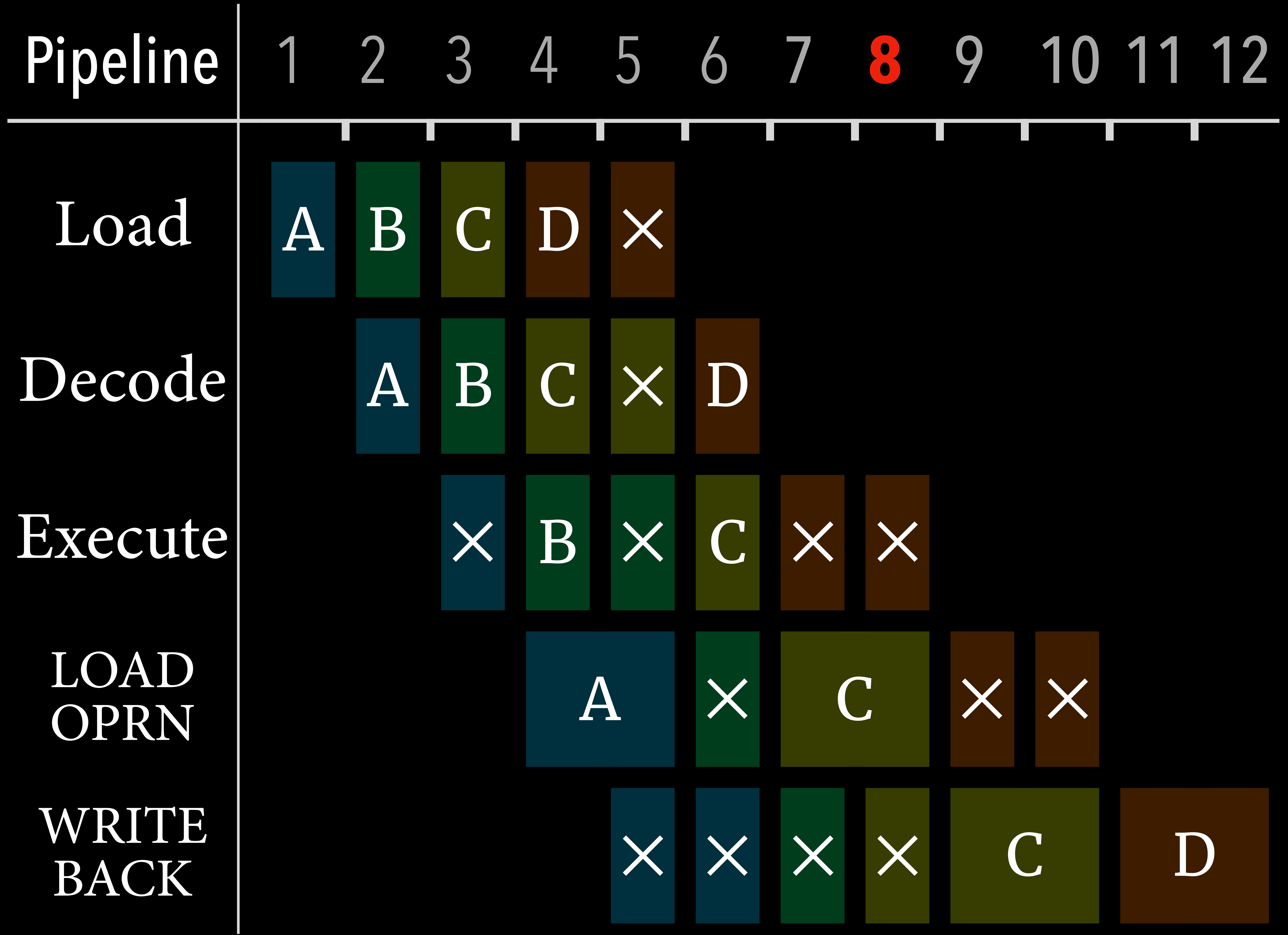
Pipeline Efficiency

The CPU runs N times faster thanks to pipelining !

Pipeline Efficiency

But it almost NEVER happens on a real CPU

Pipeline Interruptions



```
; This DEMO code shows
; different instruction
; may have different
; timing. (Assume LD
; takes longer than
; 1 Clk.)

X DW ......

............
MOV    AX, X       ; A *
INC    AX          ; B *

............
MOVSB SomeWhere ; C *
MOV    X , AX      ; D *
```

Timing Alignment Issues

Pipeline Interruptions

Pipeline	1	2	3	4	5	6	7	8
Load	A	B	C	D	…	…	…	…
Decode		A	B	C	D	…	…	…
Execute			A	B	C	D	…	…
(OPTIONAL) LOAD OPRN				A	B	C	D	…
(OPTIONAL) WRITE BK					A	B	C	D

```
; This DEMO code shows
; conditional branching
; that may interrupt
; pipeline execution.

X DW ......
Y DW ......

............
MOV AX, X         ; A *
CMP AX, Y         ; B *

JNE SomeWhere     ; C *

XOR AX, X         ; D *
```

Conditional Branches

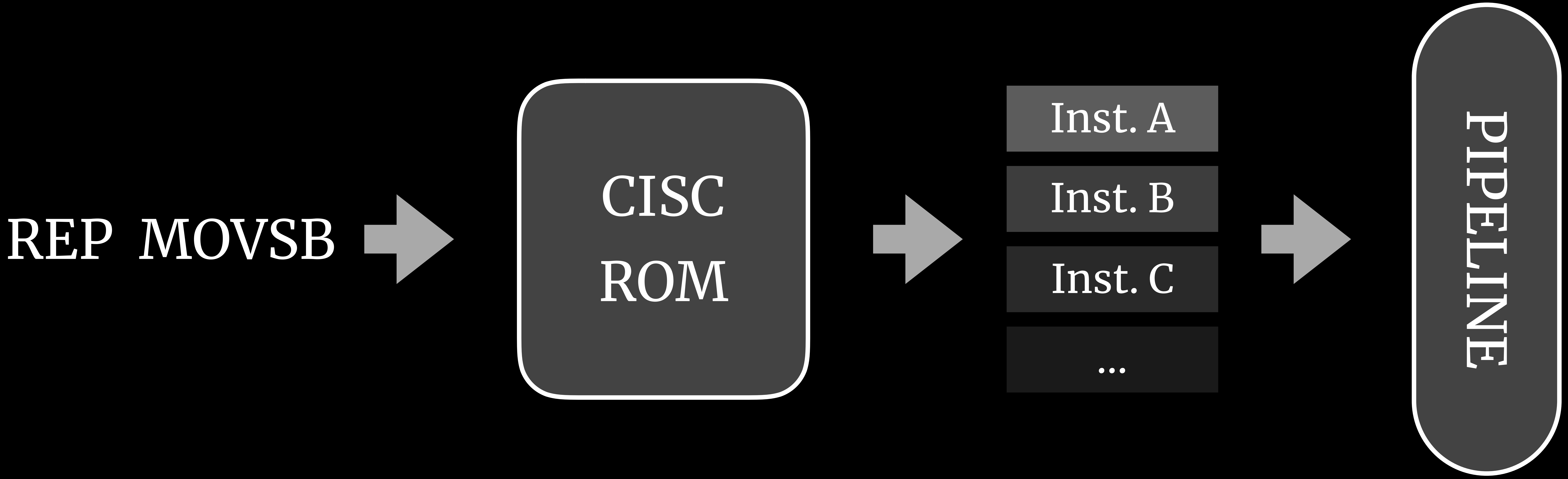
Pipeline Interruptions

Pipeline	1	2	3	4	5	6	7	8
Load	A	B	C	×	…	S	…	…
Decode		A	B	C	×	×	S	..
Execute			A	B	C	×	×	S
(OPTIONAL) LOAD OPRN				A	B	C	×	×
(OPTIONAL) WRITE BK					A	B	C	×

```
; This DEMO code shows
; conditional branching
; that may interrupt
; pipeline execution.

X DW ......
Y DW ......

............
MOV AX, X        ; A *
CMP AX, Y        ; B *

JNE SomeWhere    ; C *

XOR AX, X        ; D *
```

Conditional Branches

CISC Ins. Expansion



According to Intel Developers' Manual:

" CISC Instruction Expansion helps branch prediction "

UNSOLVED

```
APPROACH_1:
    REPNE SCASB
    JMP NEXT

APPROACH2:
    CMP CX, 0
    JZ  NEXT
    ; JMP to L2 if DF=0
L1:
    INC DI
    CMP AL, DS:[DI]
    JNE L1
    DEC CX
    JNZ L1
    JMP NEXT
L2:
    ......
```

```
DSEG SEGMENT
    STR DB "HelloWorld!","$"
DSEG ENDS
CODE SEGMENT
    ASSUME ......
START:

    ......
    MOV AL, "$"
    MOV CX, 0FFFFH
    LEA DI, STR
APPROACH_X:

......
NEXT:
    MOV AX, 0FFFFH
    SUB AX, CX
    ; AX = Length of string
CODE ENDS
END START
```

Will CISC instruction expansion be helpful ?