Superscalar, Code optimization

TD ARCHI1
Week 5
Teacher: Pirouz Bazargan-Sabet

Jorge Mendieta

25/10/23

Superscalar processors

- Rely on **spatial** parallelism
- Multiple operations running on **separate hardware** concurrently
- Achieved by **duplicating hardware resources** such as execution units and register file ports
- Requires **more** transistors

Superpipelined processors

- Rely on **temporal** parallelism
- Overlapping multiple operations on a **common hardware**
- Achieved through more **deeply pipelined execution units** with faster clock cycles
- Requires **faster** transistors

Exercise 0

c)

C code

```
for(int i=0; i < size; i++){
   v[i] = 2*v[i];
}
// size = @fin
// v = R4
```

ASM code

```
loop:
     LW    r9, 0(R4)
     SLL   r9, r9, 1
     SW    r9, 0(R4)
     ADDIU r4, r4, 4
     BNE   r4, r10, loop
```


Exercise 0

c)

C code

```
for(int i=0; i < size; i++){
   v[i] = 2*v[i];
}
// size = @fin
// v = R4
```


MIPS ASM code

```
loop:
     LW    r9, 0(R4)
     SLL   r9, r9, 1
     SW    r9, 0(R4)
     ADDIU r4, r4, 4
     BNE   r4, r10, loop
     NOP
```

0.5 lost cycle

0.5 lost cycle

1 lost cycle

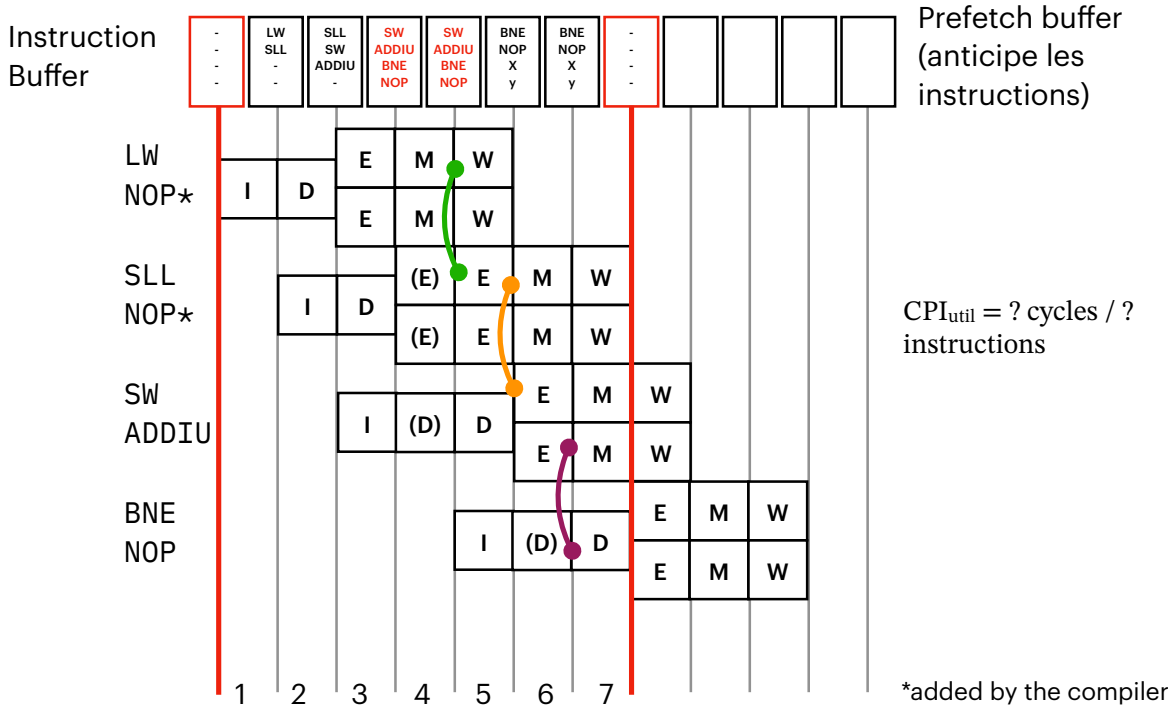
1 lost cycle

Added by "us" the programmer

3 lost cycles

Super Scalaire

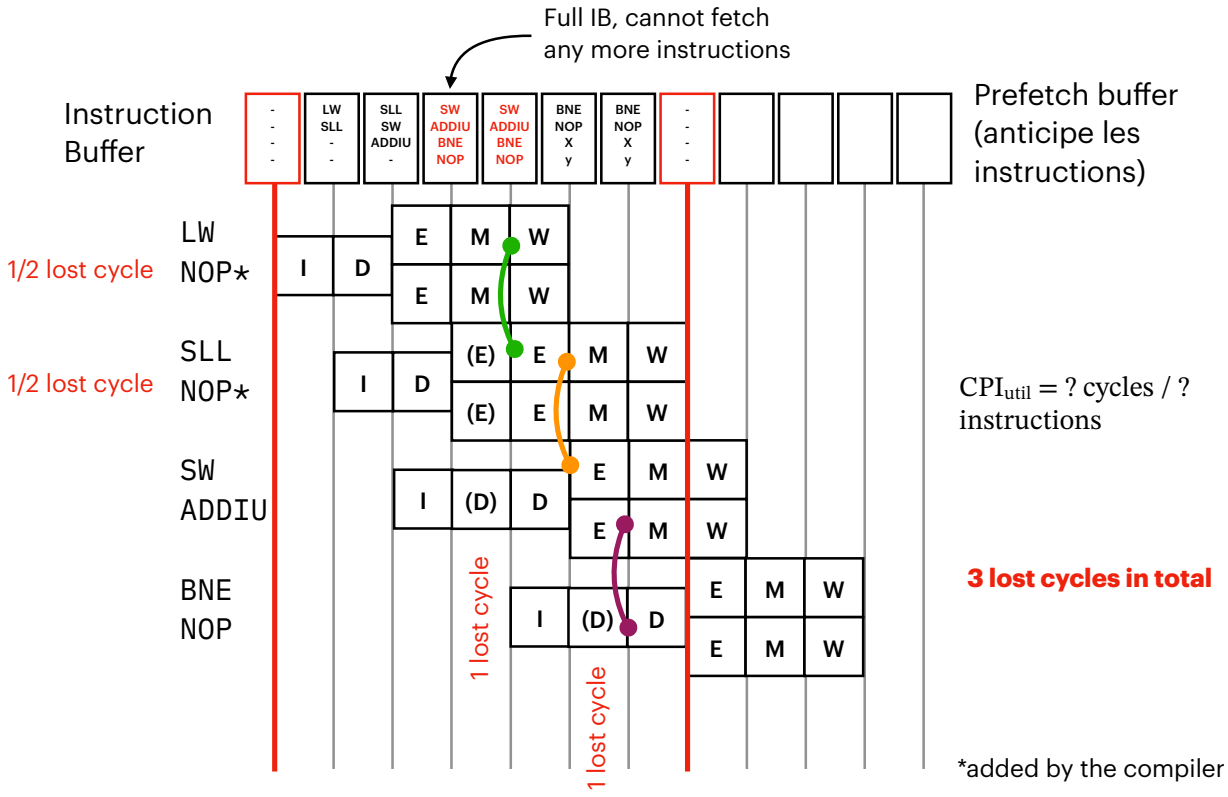
SS2



CPI_{util} = ? cycles / ? instructions

*added by the compiler

Super Scalaire



CPI_{util} = ? cycles / ? instructions

*added by the compiler

Exercise 0

c)

C code

```
for(int i=0; i < size; i++){
   v[i] = 2*v[i];
}
// size = @fin
// v = R4
```

MIPS ASM code

```
loop:
LW    r9,  0(R4)
SLL   r9,  r9, 1
SW    r9,  0(R4)
ADDIU r4,  r4, 4
BNE   r4,  r10, loop
NOP
```

0.5 lost cycle

0.5 lost cycle

1 lost cycle

1 lost cycle

3 lost cycles

$CPI_{util} = ?$

Exercise 0

d) Unrolled loop + optimizations

C code

```
for(int i=0; i < size; i+=2){
   v[i] = 2*v[i];
   v[i] = 2*v[i];
}
// size = @fin
// v = R4
```

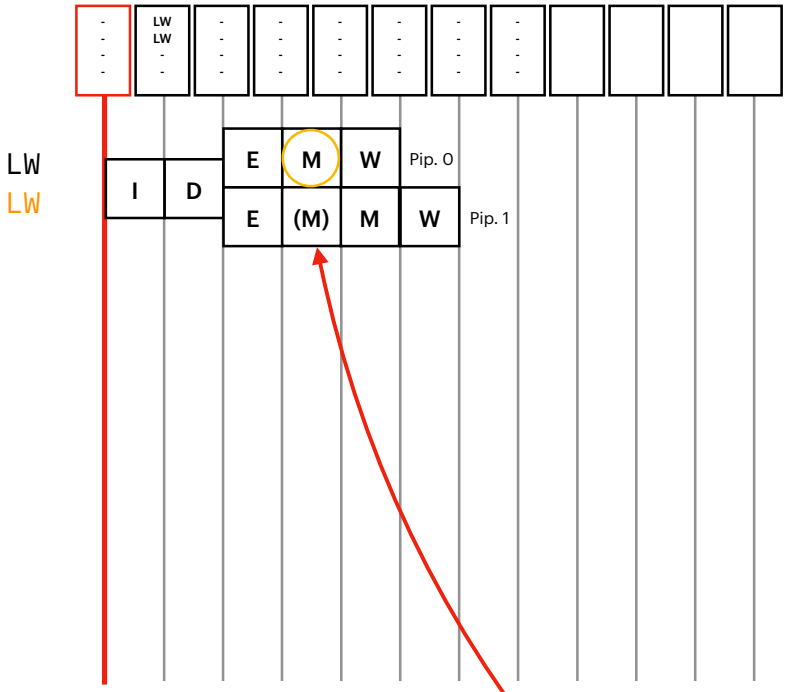
ASM code

```
loop:
LW    r9,  0(R4)
LW    r19, 4(R4)
SLL   r9,  r9,  1
SLL   r19, r19, 1
ADDIU r4,  r4, 8
SW    r9, -8(R4)
BNE   r4,  r10, loop
SW    r19, -4(r4)
```

Pay attention to the new offsets

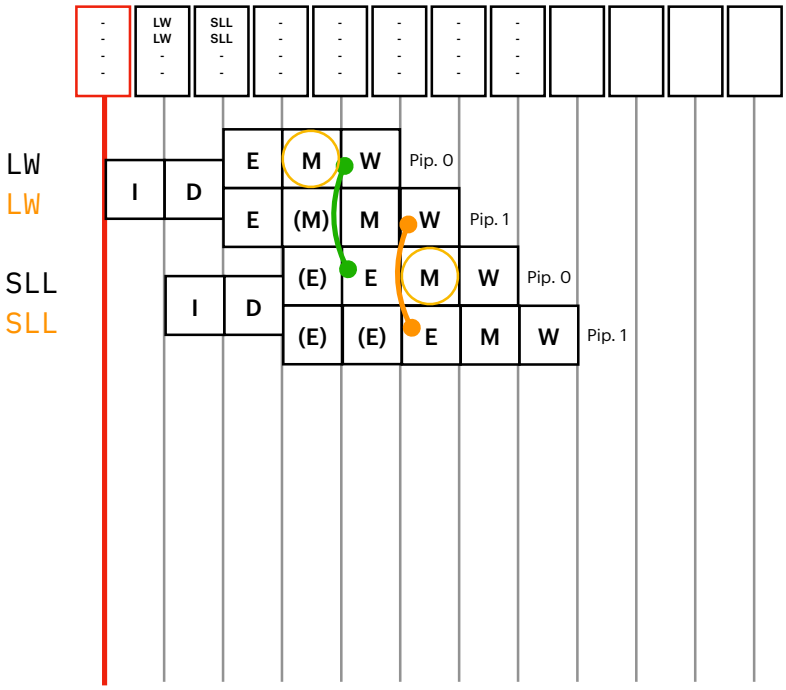
← replaces NOP

$CPI_{util} = ?$

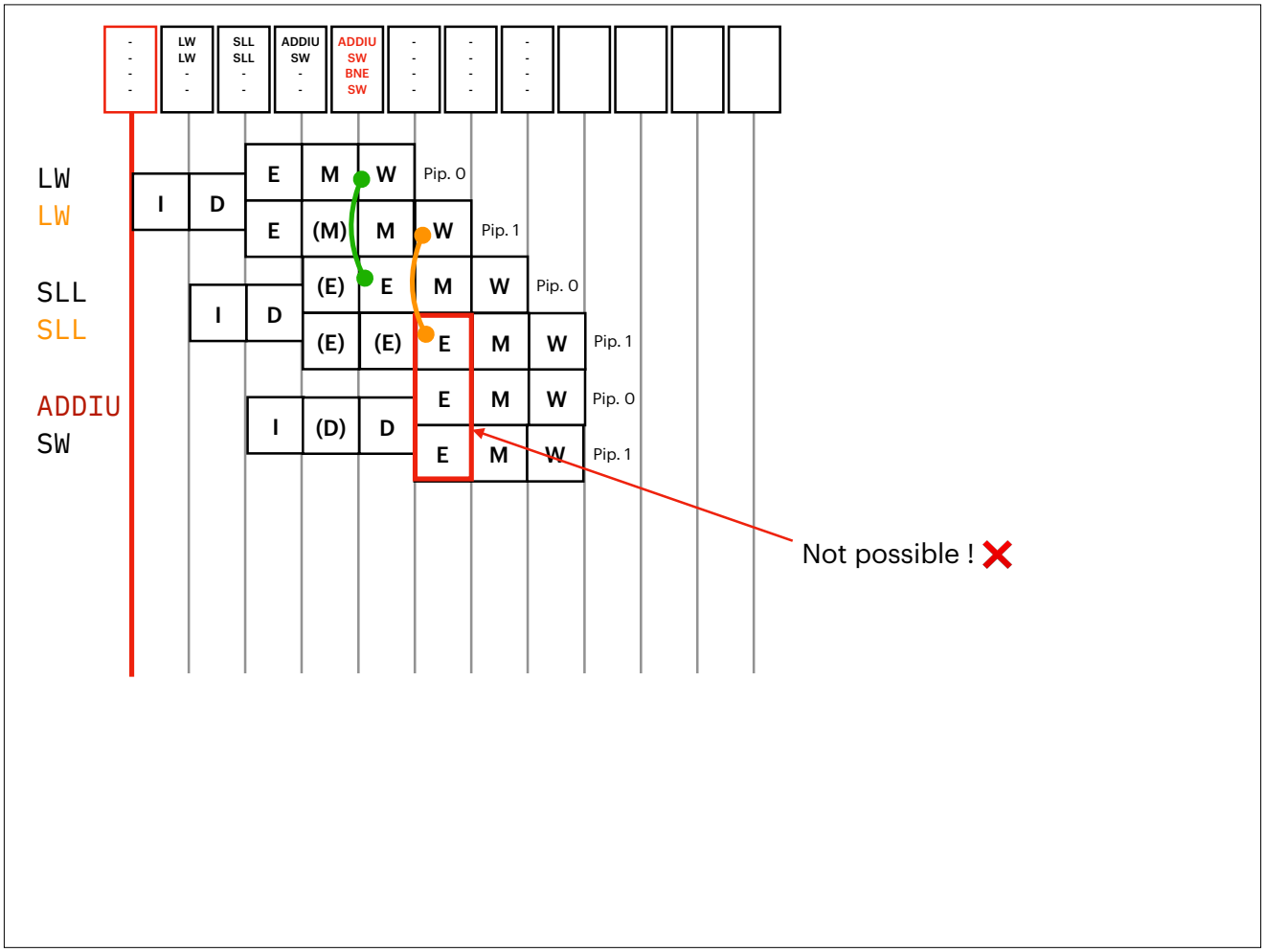


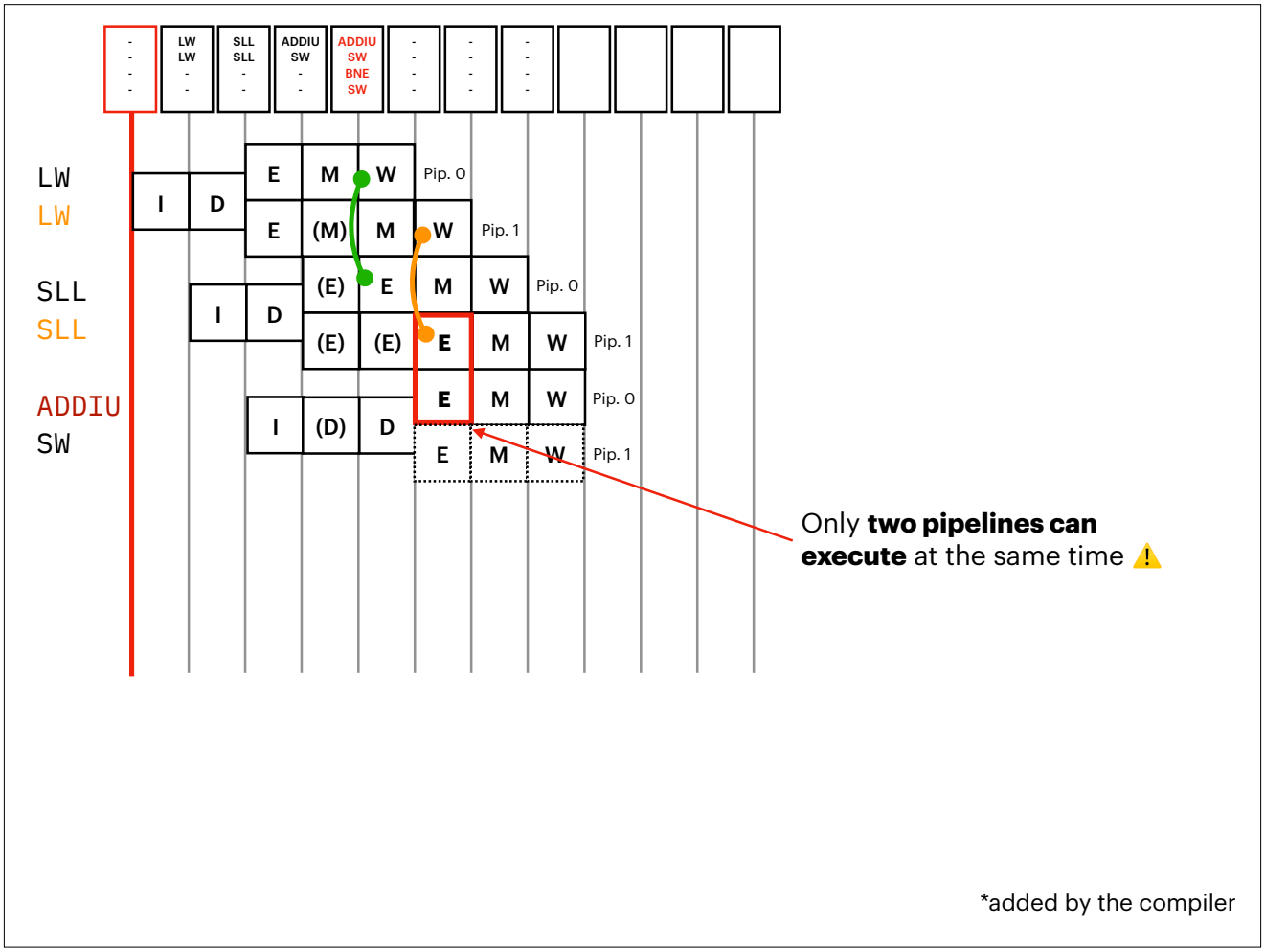
Effective memory access

Only one instruction can access the memory at a time, therefore the second pipeline needs to stall until MEM is completed ⚠

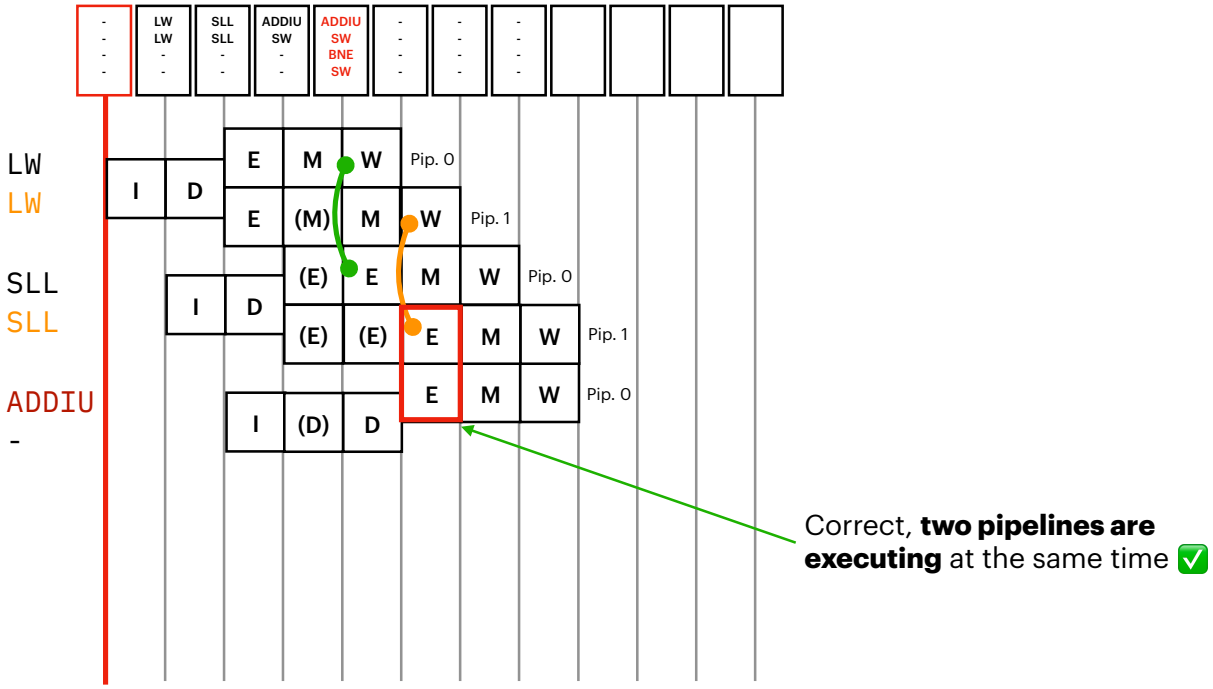


Effective memory access

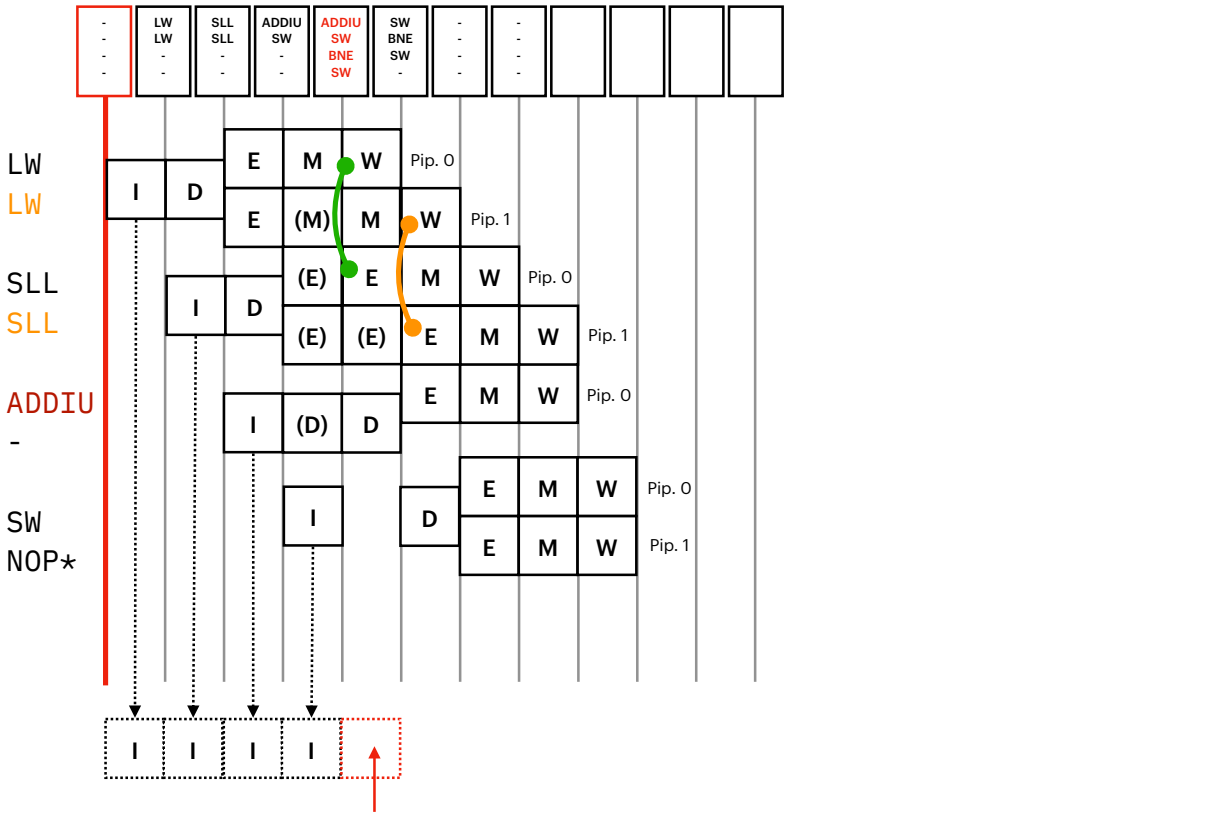




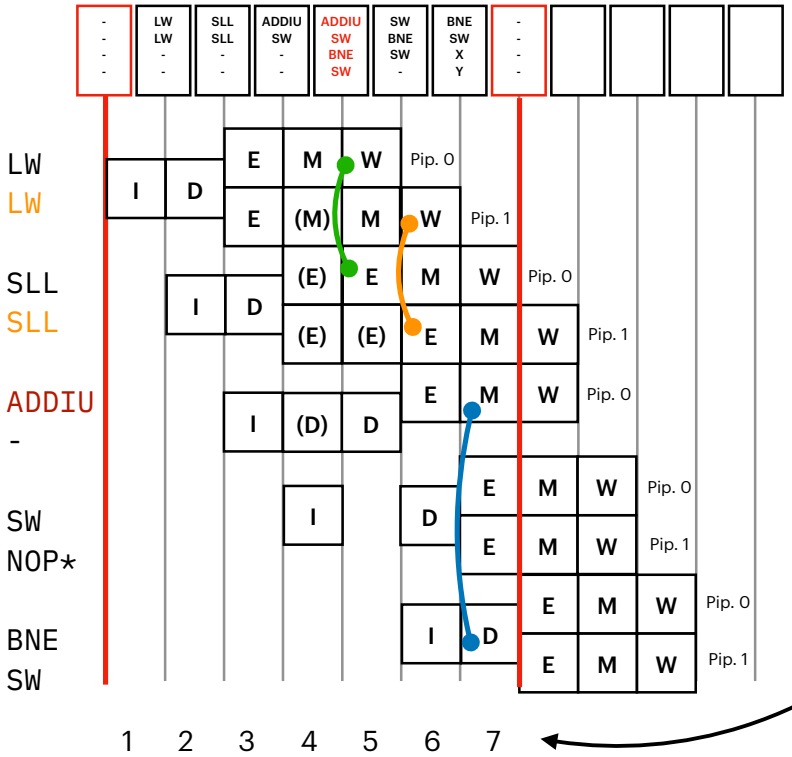
*added by the compiler



*added by the compiler



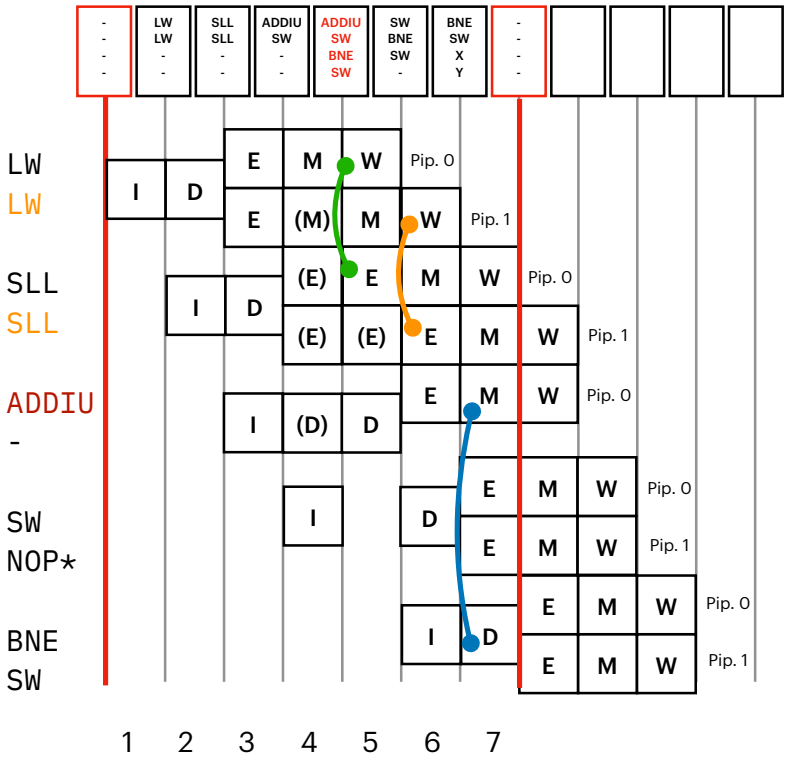
*added by the compiler



7 cycles

Number of cycles are counted since the IB is first empty until it is empty again

*added by the compiler



7 cycles
8 instructions

$\mathrm{CPI}_{\mathrm{util}} = 7 \text{ cycles} / 8 \text{ instructions}$

$\mathrm{CPI}_{\mathrm{util}} = 0.875$

*added by the compiler

Exercise 4

C code

```
for(int i=0; i < size; i+=2){
   v[i] = 2*v[i];
   v[i] = 2*v[i];
}
// size = @fin
// v = R4
```

ASM code

```
loop:
    LBU   r10, 0(r8)
    ADDU  r10, r10, r9
    LBU   r11, 0(r10)
    SB    r11, 0(r8)
    ADDIU r8, r8, 1
    BNE   r8, r12, loop
```

Exercise 4

C code

```
for(int i=0; i < size; i+=2){
   v[i] = 2*v[i];
   v[i] = 2*v[i];
}
// size = @fin
// v = R4
```

SS2 ASM code

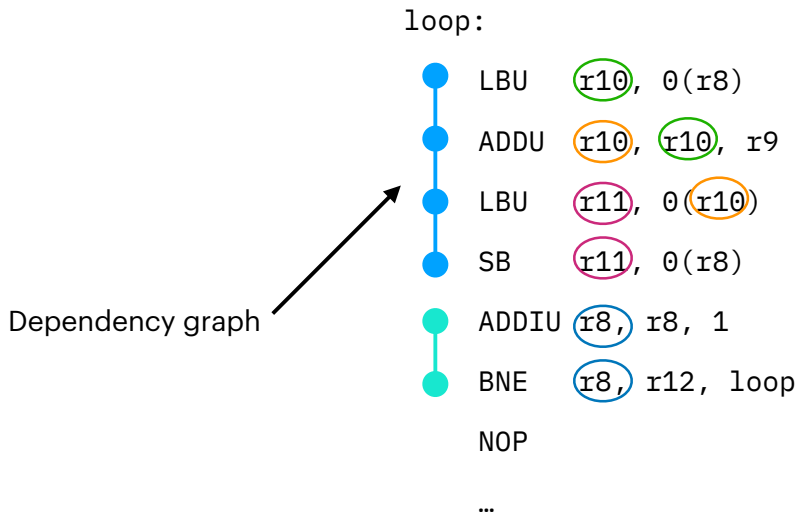
```
loop:
    LBU   r10, 0(r8)
    ADDU  r10, r10, r9
    LBU   r11, 0(r10)
    SB    r11, 0(r8)
    ADDIU r8, r8, 1
    BNE   r8, r12, loop
    NOP
    …
```

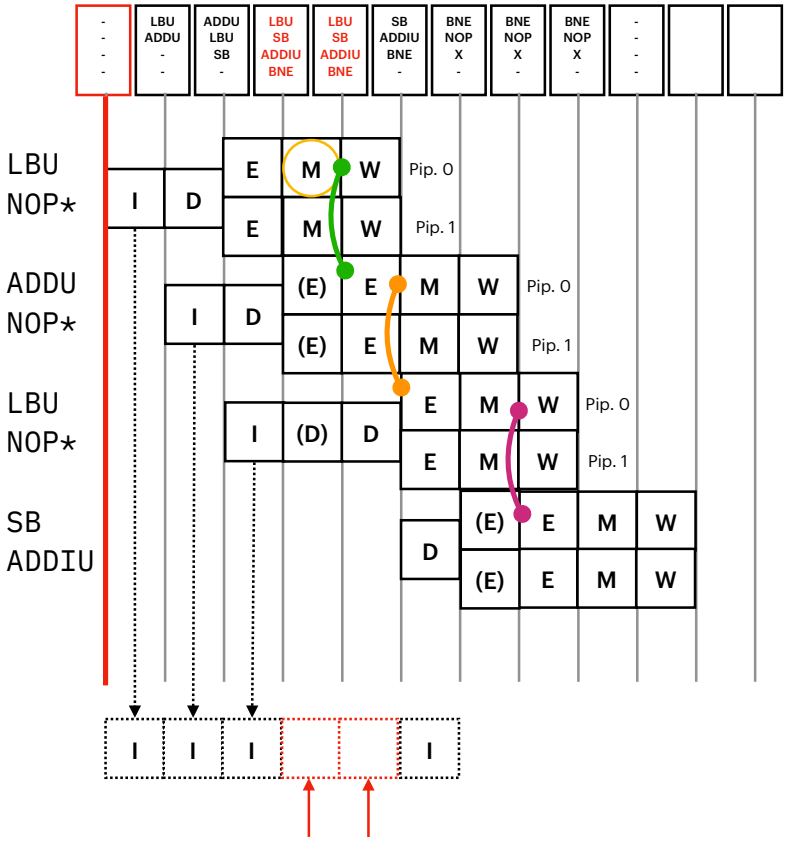
Aligned in memory ✅

Exercise 4

a)

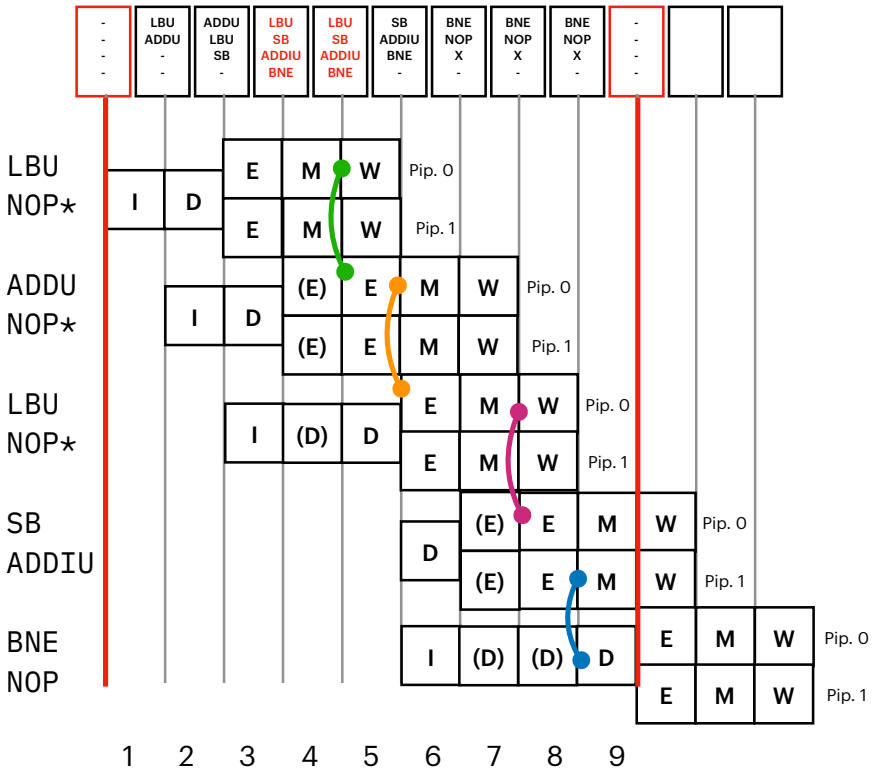
SS2 ASM code





Full IB, hence IFC stage isn't run ⚠️

*added by the compiler



9 cycles

Exercise 4

d) Optimized

SS2 ASM code

```
loop:
    LBU   r10, 0(r8)
    ADDU  r10, r10, r9
    LBU   r11, 0(r10)
    SB    r11, 0(r8)
    ADDIU r8, r8, 1
    BNE   r8, r12, loop
    NOP
    x
```

1.5 lost cycles

0.5 lost cycles

1.5 lost cycles

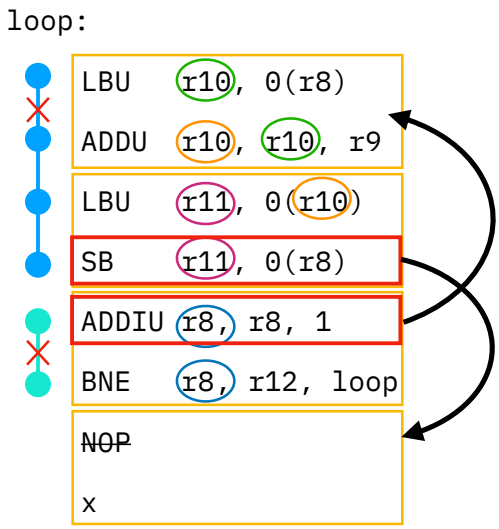
1 lost cycles

Dependency graph

Exercise 4

d) Optimized

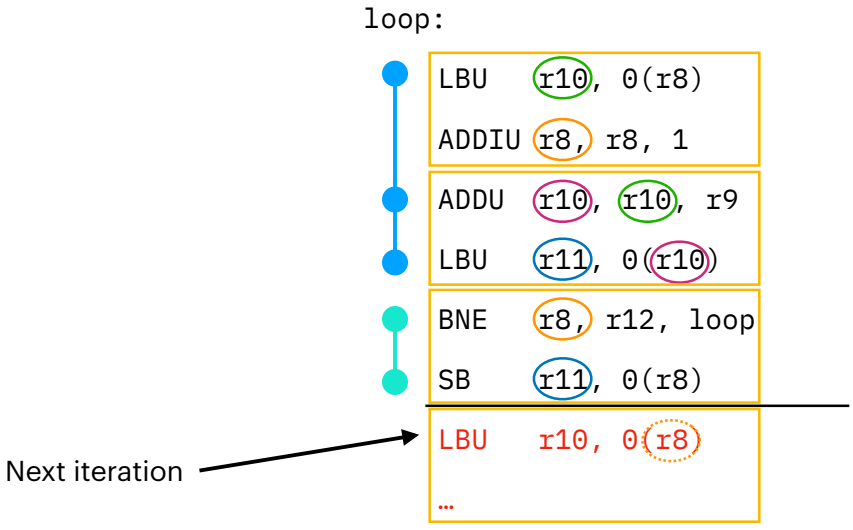
SS2 ASM code

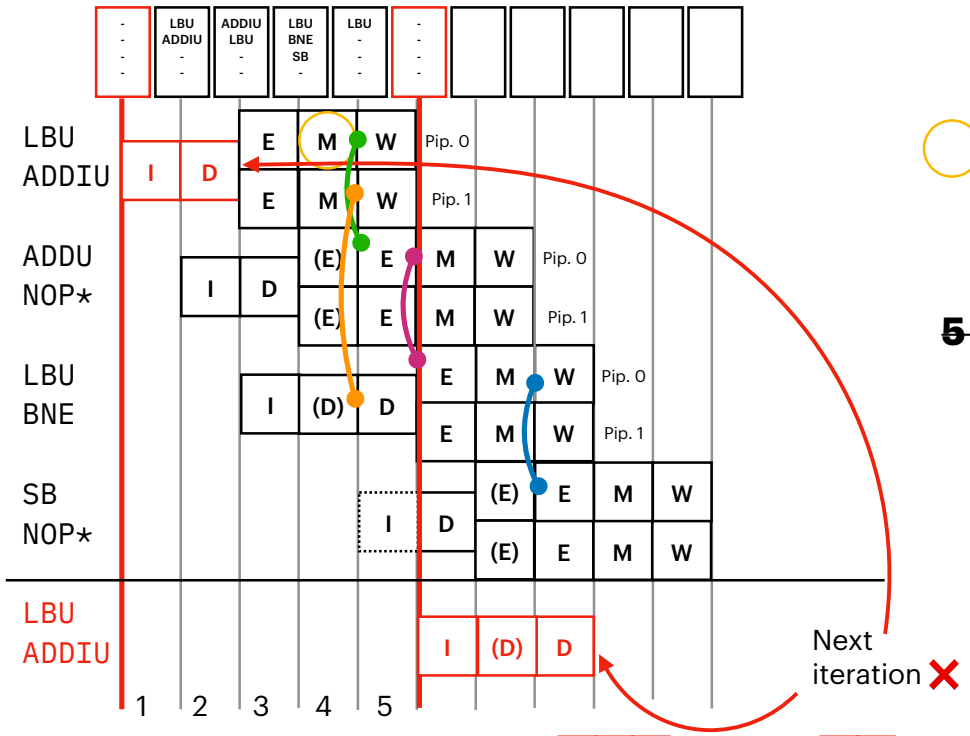


Exercise 4

d) Optimized

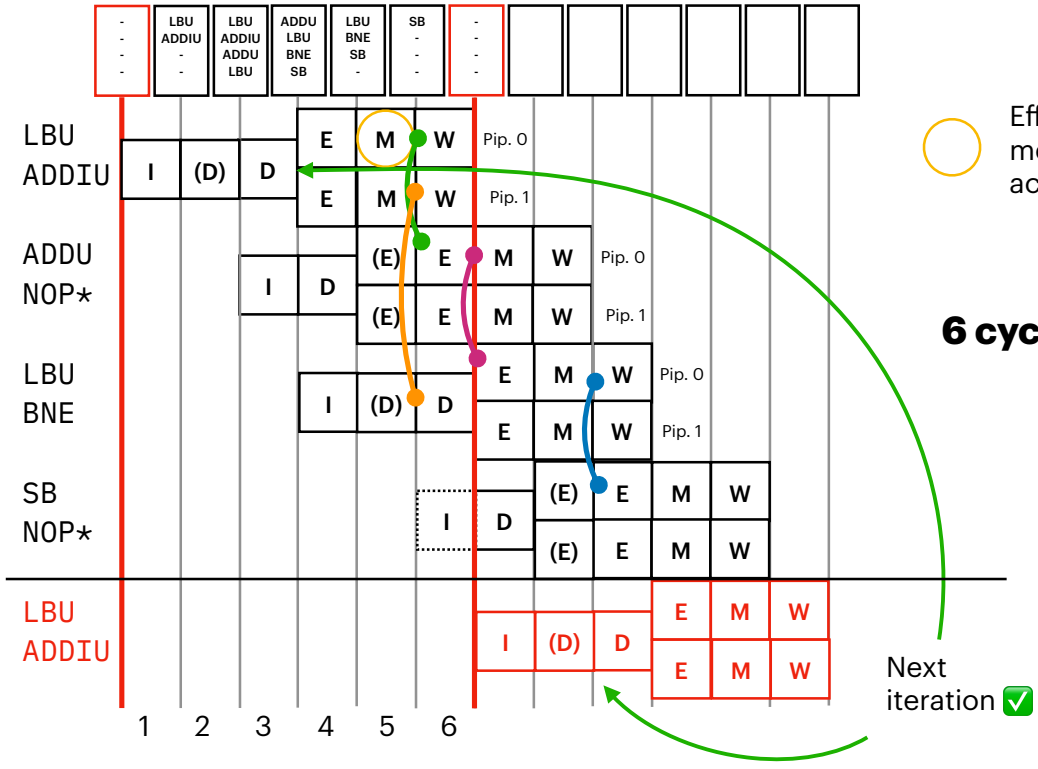
SS2 ASM code





Since the next iteration starts with | I | (D) | D | instead of | I | D |, our initial hypothesis of | I | D | is **incorrect**.

We need to restart the diagram with the new assumption that the first cycle starts with | I | (D) | D | instead of | I | D |



Our new hypothesis of the first iteration starting with a stall cycle in the decode stage | I | (D) | D | is **correct**, since the next iteration repeats with | I | (D) | D |