CS614 Advanced Compilers

03.03.25

INSTRUCTION SCHEDULING

Pipelined architecture makes compiler's job interesting.

	i	i+1	i+2	i+3	i+4
1	IF				
2	ID	IF			
3	OF	ID	IF		
4	EX	OF	ID	IF	
5	WB	EX	OF	ID	IF
6		WB	EX	OF	ID
7			WB	EX	OF
8				WB	EX
9					WB

9 cycles instead of 25 (5×5).

Stalls possible due to control & data hazards.

↳ Some can be reduced due to features such as operand-forwarding from write buffers.

① Data dependency — H/w : prefetch — Compiler: reorder

② Control dependency — H/w : predict with a few bits — Compiler: speculate

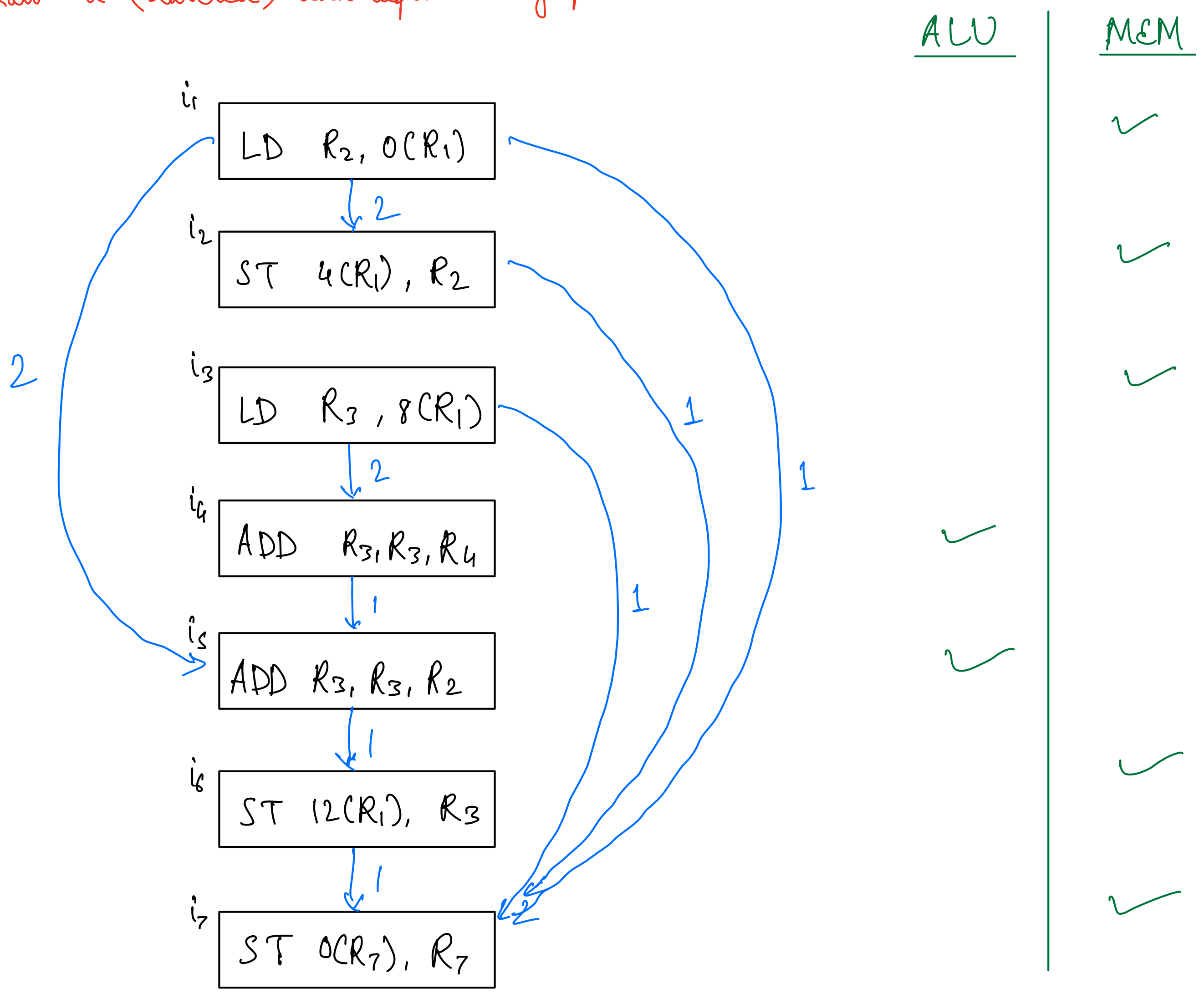
Basic-Block Scheduling

LD: 2 cycles ; rest 1 cycle
↳ next ST can start after 1 cycle

- Say we have separation of ALU and MEM execution.

OP dst, src

① Draw a (labeled) data-dependence graph.



② Choose a prioritized topological order.

$i_3 \rightarrow i_4 \rightarrow i_5 \rightarrow i_6 \rightarrow i_7$: 6 cycles

$i_1 \rightarrow i_5 \rightarrow i_6 \rightarrow i_7$: 5 cycles

$i_1 \rightarrow i_2 \rightarrow i_7$: 4 cycles

⋮

Heuristic: Length of critical path

$i_3 \rightarrow i_1 \rightarrow i_2 \rightarrow i_6 \rightarrow i_7$: MEM

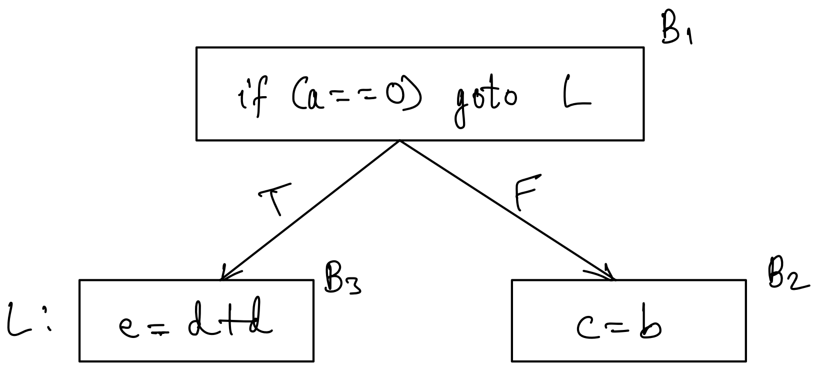
$i_4 \rightarrow i_5$: ALU

	ALU	MEM
1:		LD R3, 8(R1)
2:		LD R2, 0(R1)
3:	ADD R3, R3, R4	
4:	ADD R3, R3, R2	
5:		ST 4(R1), R2
6:		ST 12(R1), R3
7:		ST 0(R7), R7

7 cycles

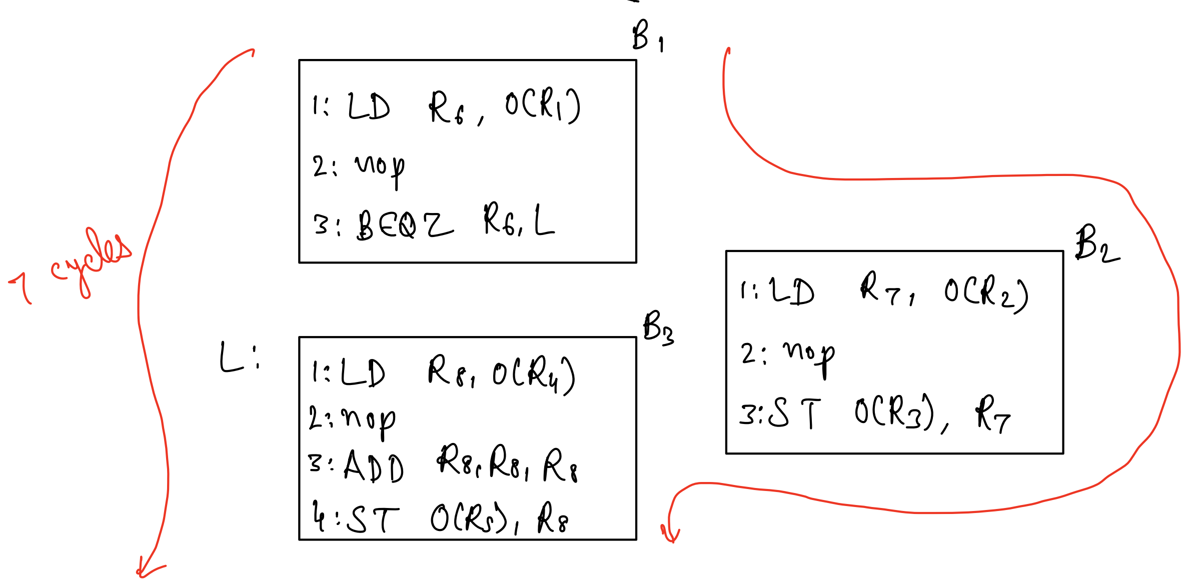
06.03.25

Global Scheduling



RA	
a	R1
b	R2
c	R3
d	R4
e	R5

⇓ (ISel + RA)



B1:
```
1: LD R6, 0(R1)
2: nop
3: BEQZ R6, L
```

B2:
```
1: LD R7, 0(R2)
2: nop
3: ST 0(R3), R7
```

L: B3:
```
1: LD R8, 0(R4)
2: nop
3: ADD R8, R8, R8
4: ST 0(R5), R8
```

10 cycles

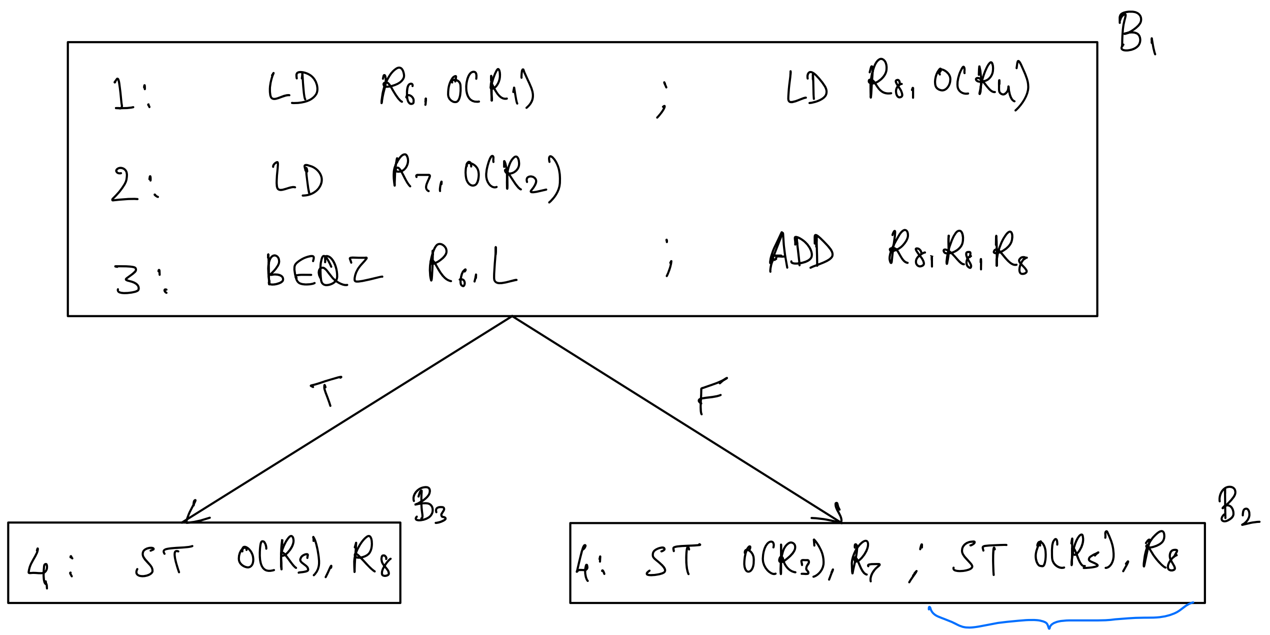
// Local Instruction Schedule

Observations:-

1. No data dependence among B_1, B_2 & B_3.
2. B_2 is control dependent on B_1.
3. B_3 can be scheduled in parallel with B_1.
4. B_1 should be prioritized before B_3.
5. LD in B_2 can be moved to B_1, ST cannot.

instruction reordering across BBs

Let's say our hardware allows issuing two instructions simultaneously:



B_1
```
1:   LD   R6, 0(R1)      ;   LD   R8, 0(R4)
2:   LD   R7, 0(R2)
3:   BEQZ R6, L          ;   ADD  R8, R8, R8
```
T → B_3
```
4:   ST   0(R5), R8
```
F → B_2
```
4:   ST   0(R3), R7  ;  ST  0(R5), R8
```

duplicated to save one cycle (edge deleted between B_2 & B_3)

4 cycles

Considerations:-

1. Data dependencies
2. Control dependencies
3. Pipelined h/w
4. Parallelism in h/w
5. Heterogeneity, distributed

Option:-

out of order execution, instrcn reordering
speculative execution, code motion / duplication
parallelize / vectorize
communication optimization