Exe 2 VLIW: Architecture

- Consider the program be executed on a **3-issue** VLIW MIPS (Very Long Instruction Word) architecture with **3 fully pipelined functional** units

- **Integer ALU** with **1** cycle latency to **next Integer/FP** and **2** cycle latency to **next Branch**

- **Memory** Unit with **3** cycle latency

- **Floating** Point Unit with **3** cycle latency (each **FPU** can complete one add or one multiply per clock cycle)

- Branch completed with **1 cycle delay slot** (branch solved in ID stage)

Exe 2 VLIW: schedule

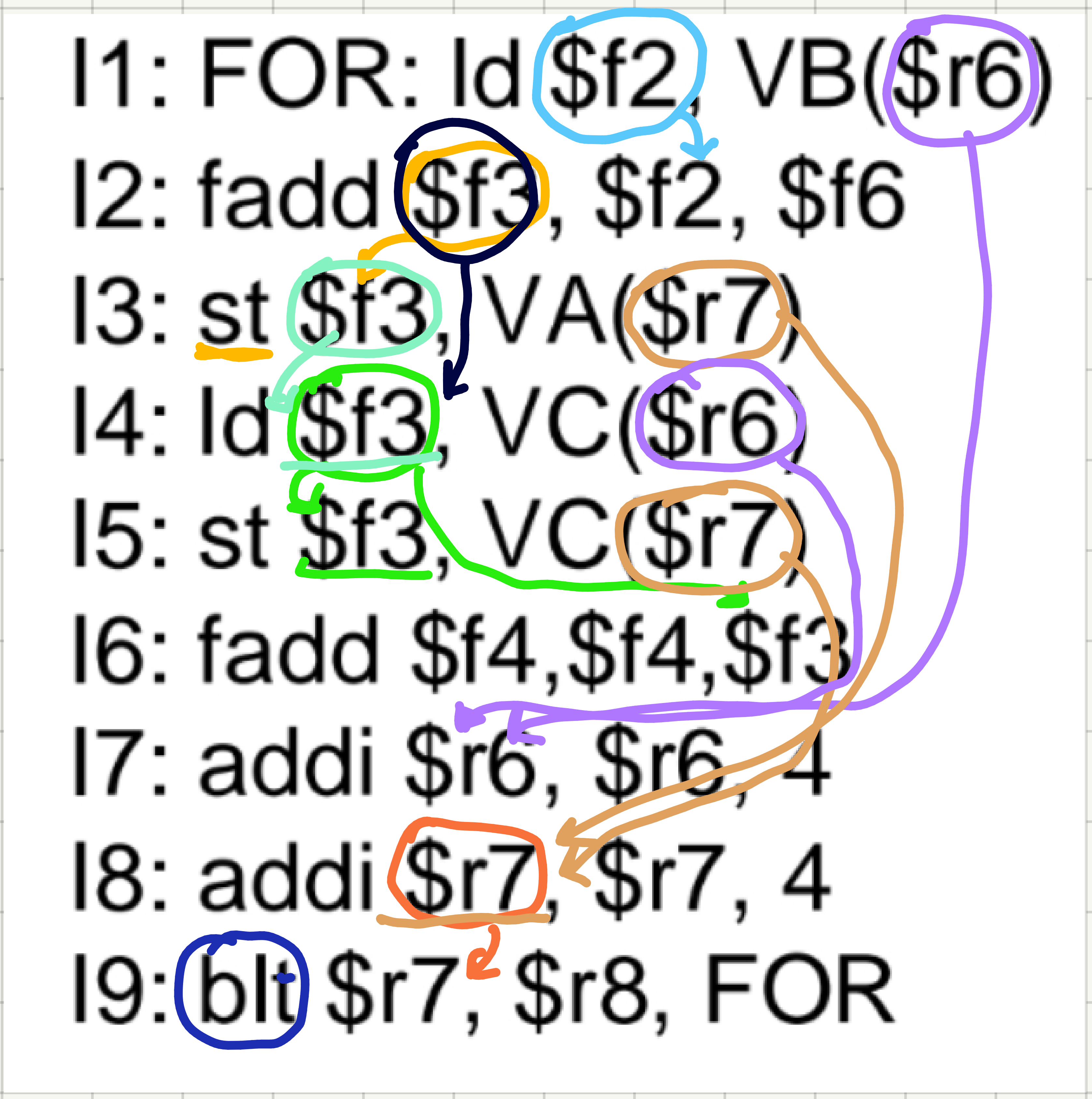
- Considering **one iteration** of the loop
- schedule the assembly code for the 3-issue VLIW machine in the following table by using the **list-based scheduling**
- **Do not** use neither software pipelining nor loop unrolling nor modifying loop indexes.
- Please do not need to write in NOPs (can leave blank).

Exe 2 VLIW: the code

Assembly Code:

```
FOR:  ld     $f2, VB($r6)
      fadd   $f3, $f2, $f6
      st     $f3, VA($r7)
      ld     $f3, VC($r6)
      st     $f3, VC($r7)
      fadd   $f4,$f4,$f3
      addi   $r6, $r6, 4
      addi   $r7, $r7, 4
      blt    $r7, $r8, FOR
```

CONFLICTS



```
I1: FOR: ld $f2, VB($r6)
I2: fadd $f3, $f2, $f6
I3: st $f3, VA($r7)
I4: ld $f3, VC($r6)
I5: st $f3, VC($r7)
I6: fadd $f4,$f4,$f3
I7: addi $r6, $r6, 4
I8: addi $r7, $r7, 4
I9: blt $r7, $r8, FOR
```

RAW WAW

RAW WAR

RAW x 2

RAW

WAR x 2

RAW x2

CNTRL

Exe 2 VLIW: schedule

```
FOR: ld    $f2, VB($r6) ✓
     fadd  $f3, $f2, $f6 ✓
     st    $f3, VA($r7) ✓
     ld    $f3, VC($r6) ✓
     st    $f3, VC($r7) ✓
     fadd  $f4,$f4,$f3 ✓
     addi  $r6, $r6, 4 ✓
     addi  $r7, $r7, 4 ✓
     blt   $r7, $r8, FOR ✓
```

	Integer ALU(1/2 b)	Memory Unit(3cc)	FPU(3cc)
C1		ld $F2, VB($r6) 1	
C2		2	
C3		3	
C4			fadd $F3, $F2, $f6 1
C5			2
C6			3
C7		st $f3, VA($r7)	
C8	addi $r6,$r6,4	ld $F3, VC($r6) 1	
C9		2	
C10		3	
C11	addi $r7,$r7,4 1	st $F3, VC($r7)	fadd $F4, $F4, $F3
C12	2		
C13	blt $r7, $r8, FOR		
C14			
C15			

Exe 2 VLIW: schedule

```
FOR: ld    $f2, VB($r6)   ✓
     fadd  $f3, $f2, $f6  ✓
     st    $f3, VA($r7)   ✓
     ld    $f3, VC($r6)   ✓
     st    $f3, VC($r7)   ✓
     fadd  $f4,$f4,$f3    ✓
     addi  $r6, $r6, 4    ✓
     addi  $r7, $r7, 4    ✓
     blt   $r7, $r8, FOR  ✓
```

	Integer ALU(1/2 b)	Memory Unit(3cc)	FPU(3cc)
C1	nop	ld $f2, VB($r6) 1	nop
C2	nop	nop 2	nop
C3	nop	nop 3	nop
C4	nop	nop	fadd $f3, $f2, $f6 1
C5	nop	nop	nop 2
C6	nop	nop	nop 3
C7	nop	st $f3, VA($r7)	nop
C8	addi $r6,$r6,4	ld $f3, VC($r6) 1	nop
C9	nop	nop 2	nop
C10	nop	nop 3	nop
C11	addi $r7,$r7,4 1	st $f3, VC($r7)	fadd $f4, $f4, $f3
C12	nop 2	nop	nop
C13	blt $r7, $r8, FOR	nop	nop
C14	nop	nop	nop
C15	nop	nop	nop

Exe 1 VLIW: Architecture

- Consider the program be executed on a **3-issue** VLIW MIPS (Very Long Instruction Word) architecture with **3 fully pipelined functional** units
- **Integer ALU** with **1** cycle latency
- **Memory Unit** with **2** cycle latency
- **Floating Point Unit** with **3** cycle latency
- Branch solved in EXE stage,

  no early evaluation

Exe VLIW.1: schedule

- Considering **one iteration** of the loop
- schedule the assembly code for the 3-issue VLIW machine in the following table by using the **list-based scheduling** with **ASAP**
- Calculate the **performance** (FLOPs per cycle)

- **Do not** use neither software pipelining nor loop unrolling nor modifying loop indexes
- Please do not need to write in NOPs (can leave blank)

Exe VLIW.2: schedule

- **Unroll** the loop by one iteration (so two iterations of the original loop are performed for every branch in the new assembly code)
- You only need to worry about the steady-state code in the core of the loop (no epilogue or prologue)
- schedule the assembly code for the 3-issue VLIW machine in the following table by using the **list-based scheduling** with **ASAP**
- Calculate the **performance** (FLOPs per cycle)

- **Do not** use software pipelining
- Please do not need to write in NOPs (can leave blank)

Exe VLIW.1: the code

C Code:

```
for(int i=0; i<N; i++) {
    C[i] = A[i]*A[i] + B[i];
}
```

Assembly Code:

```
loop:  ld     f1, 0(r1)
       ld     f2, 0(r2)
       fmul   f1, f1, f1
       fadd   f1, f1, f2
       st     f1, 0(r3)
       addi   r1, r1, 4
       addi   r2, r2, 4
       addi   r3, r3, 4
       bne    r3, r4, loop
```

Exe VLIW.1: schedule

```
loop: ld f1, 0(r1) ✓
      ld f2, 0(r2) ✓
      fmul f1, f1, f1 ✓
      fadd f1, f1, f2 ✓
      st f1, 0(r3) ✓
      addi r1, r1, 4 ✓
      addi r2, r2, 4 ✓
      addi r3, r3, 4 ✓
      bne r3, r4, loop ✓
```

"READ AFTER READ IS NOT A DEPENDENCE

FLOPS/cc = 2/10

	Integer ALU (1 cc)	Memory Unit (2 cc)	FPU (3 cc)
C1	addi r1,r1,4 1	ld F1, 0(r1) 1	
C2	addi r2,r2,4	ld F2,0(r2) 2	
C3			Fmul F1,F1,F1 1
C4			2
C5			3
C6			Fadd F1,F1,F2 1
C7	READ r3 ON IS BEFORE WRITE		2
C8			3
C9	addi r3,r3,4 1	st F1, 0(r3)	
C10	bne r3,r4,loop		
C11			
C12			
C13			
C14			
C15			

Exe 1 VLIW: Architecture

- Consider the program be executed on a **3-issue** VLIW MIPS (Very Long Instruction Word) architecture with **3 fully pipelined functional** units
- **Integer ALU** with **1** cycle latency
- **Memory Unit** with **2** cycle latency
- **Floating Point Unit** with **3** cycle latency
- Branch solved in EXE stage,

  <u>no early evaluation</u>

Exe VLIW.1: schedule

- Considering **one iteration** of the loop
- schedule the assembly code for the 3-issue VLIW machine in the following table by using the **list-based scheduling** with **ASAP**
- Calculate the **performance** (FLOPs per cycle)

- **Do not** use neither software pipelining nor loop unrolling nor modifying loop indexes
- Please do not need to write in NOPs (can leave blank)

Exe VLIW.2: schedule

- **Unroll** the loop by one iteration (so two iterations of the original loop are performed for every branch in the new assembly code)
- You only need to worry about the steady-state code in the core of the loop (no epilogue or prologue)
- schedule the assembly code for the 3-issue VLIW machine in the following table by using the **list-based scheduling** with **ASAP**
- Calculate the **performance** (FLOPs per cycle)

- **Do not** use software pipelining
- Please do not need to write in NOPs (can leave blank)

Exe VLIW.2: the code

C Code:

```
for(int i=0; i<N; i+=2) {
    C[i] = A[i]*A[i] + B[i];
    C[i+1] = A[i+1]*A[i+1] + B[i+1];
}
```

Assembly Code:

```
loop:   ld      f1, 0(r1)
        ld      f3, 4(r1)
        ld      f2, 0(r2)
        ld      f4, 4(r2)
        fmul    f1, f1, f1
        fmul    f3, f3, f3
        fadd    f1, f1, f2
        fadd    f3, f3, f4
        st      f1, 0(r3)
        st      f3, 4(r3)
        addi    r1, r1, 8
        addi    r2, r2, 8
        addi    r3, r3, 8
        bne     r3, r4, loop
```

Exe VLIW.2: schedule

```
loop: ld   f1, (r1)     ✓
      ld   f3, 4(r1)    ✓
      ld   f2, 0(r2)    ✓
      ld   f4, 4(r2)    ✓
      fmul f1, f1, f1   ✓
      fmul f3, f3, f3   ✓
      fadd f1, f1, f2   ✓
      fadd f3, f3, f4   ✓
      st   f1, 0(r3)    ✓
      st   f3, 4(r3)    ✓
      addi r1, r1, 8    ✓
      addi r2, r2, 8    ✓
      addi r3, r3, 8    ✓
      bne  r3, r4, loop ✓
```

	Integer ALU (1 cc)	Memory Unit (2 cc)	FPU (3 cc)
C1		ld f1, (r1)	
C2	addi r1, r1, 8 *	ld f3, 4(r1) 1	
C3		ld f2, 0(r2) 1 2	fmul f1, f1, f1 1
C4	addi r2, r2, 8	ld f4, 4(r2) 2 1	fmul f3, f3, f3 1 2
C5		2	2 3
C6			1 fadd f1, f1, f2 3
C7			2 fadd f3, f3, f4
C8			3
C9		st f1, 0(r3)	
C10	addi r3, r3, 8 1	st f3, 4(r3)	
C11	bne r3, r4, loop		
C12			
C13			
C14			
C15			

*r1 READ ON 1st HALF OF C2, WRITTEN IN 2nd