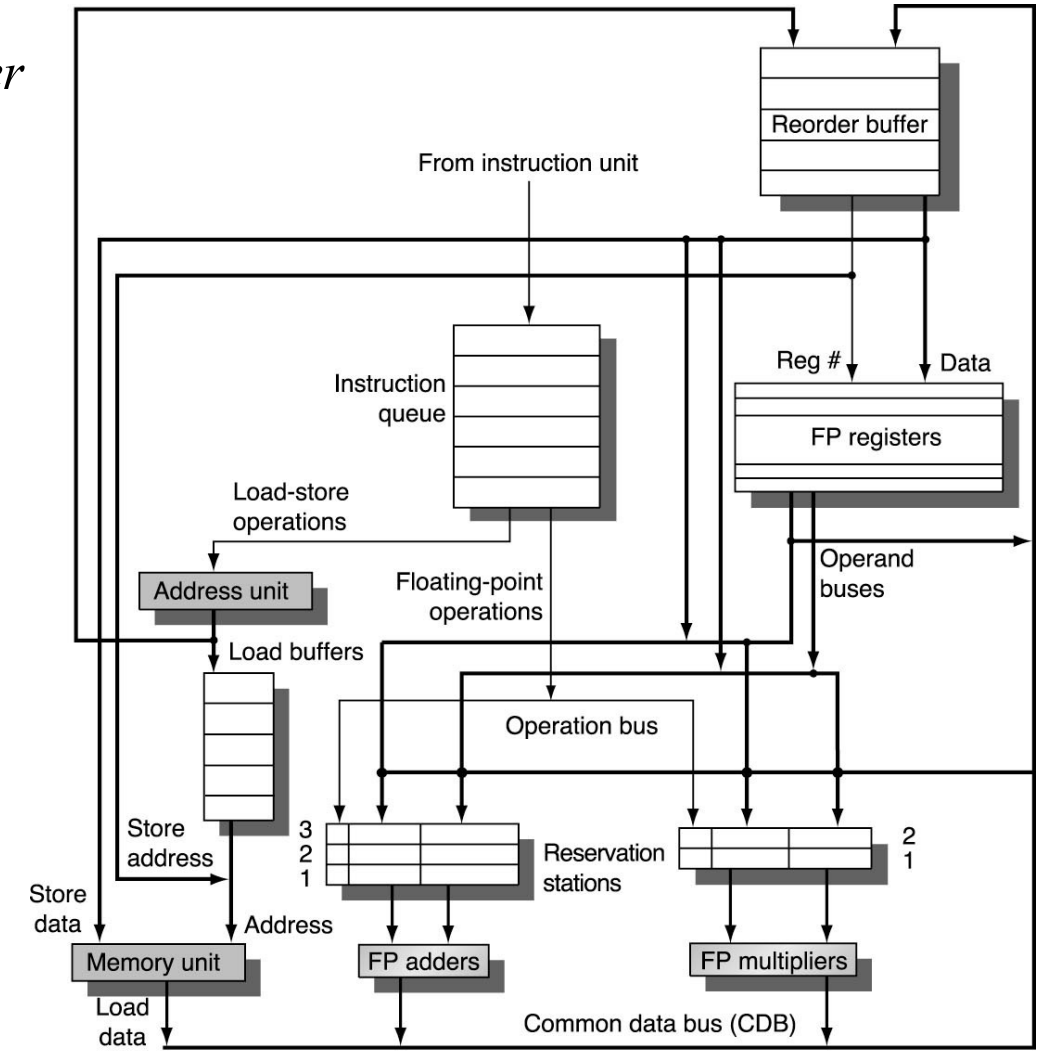
The Reorder Buffer

Hardware Speculative Execution

- Need HW buffer for results of uncommitted instructions: *reorder buffer*
  - Reorder buffer can be operand source
  - Once operand commits, result is found in register
  - 3 fields: instr. type, destination, value
  - Use reorder buffer number instead of reservation station
  - Instructions commit in order
  - As a result, its easy to undo speculated instructions on mispredicted branches or on exceptions



Hardware Speculative Execution

- Need HW buffer for results of uncommitted instructions: *reorder buffer*
  - 3 fields:
    - instruction type
    - destination
    - value
  - once operand commits, result is found in register file
  - reservation station points to a ROB entry for pending source operand
  - flush ROB on a mispredicted branch
  - instructions are committed in-order from the ROB
    - handles precise exceptions

~~Four~~/Five Steps of Speculative ROB/Tomasulo Algorithm

1. Dispatch—get instruction from FP Op Queue

   If reservation station and reorder buffer slot free, issue instr & send operands & reorder buffer no. for destination.

2. Issue—wait on operands

   When both operands ready then execute; if not ready, watch CDB for result; when both in reservation station, procede to execute

3. Execute —

4. Write result—finish execution (WB)

   Write on Common Data Bus to all awaiting FUs & reorder buffer; mark reservation station available.

5. Commit—update register with reorder result

   When instr. at head of reorder buffer & result present, update register with result (or store to memory) and remove instr from reorder buffer.

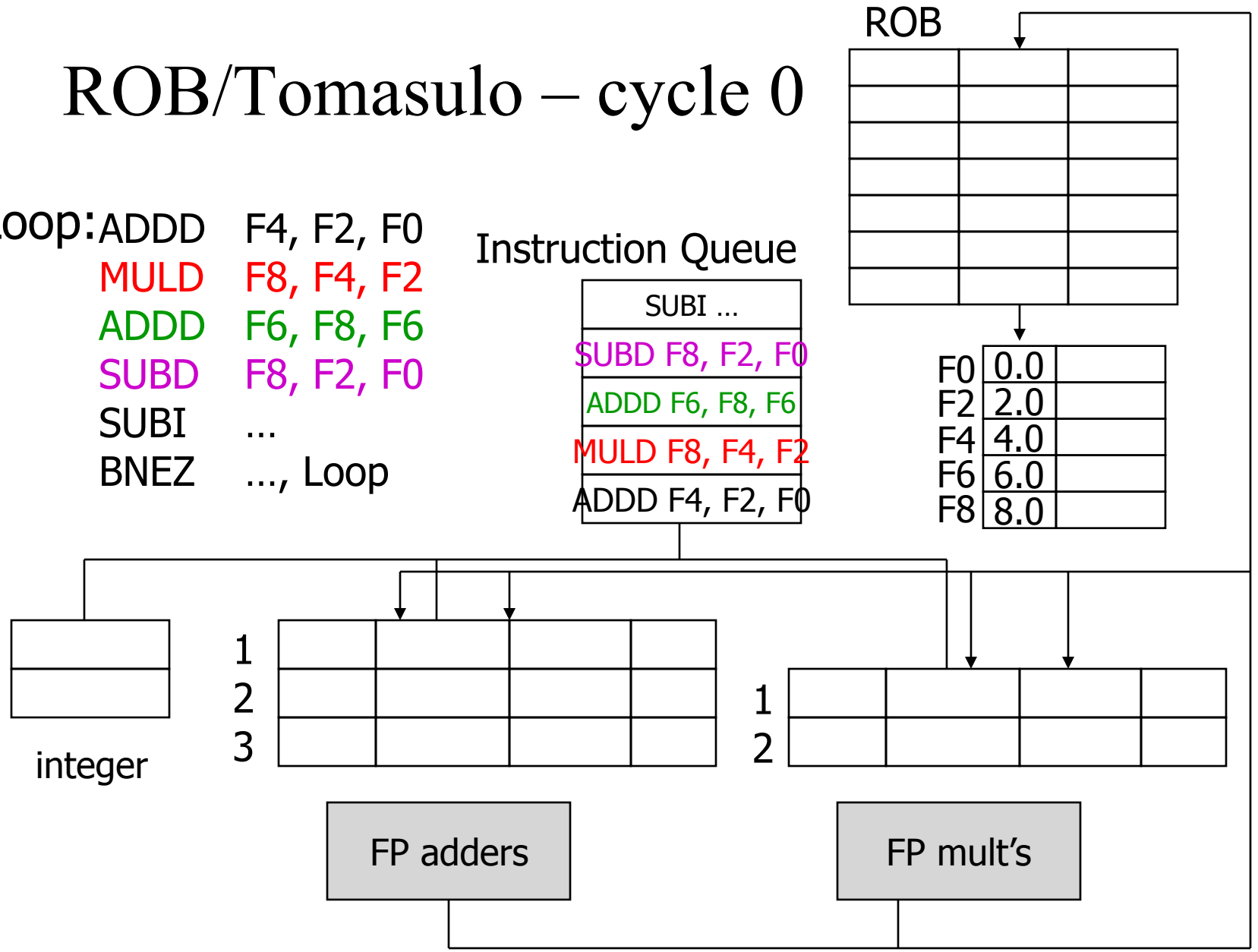
HW Speculative Execution

- The re-order buffer and in-order commit allow us to flush the speculative instructions from the machine when a misprediction is discovered.

- ROB is another possible source of operands
- ROB can provide precise exception in an out-of-order machine
- ROB allows us to ignore exceptions on speculative code.

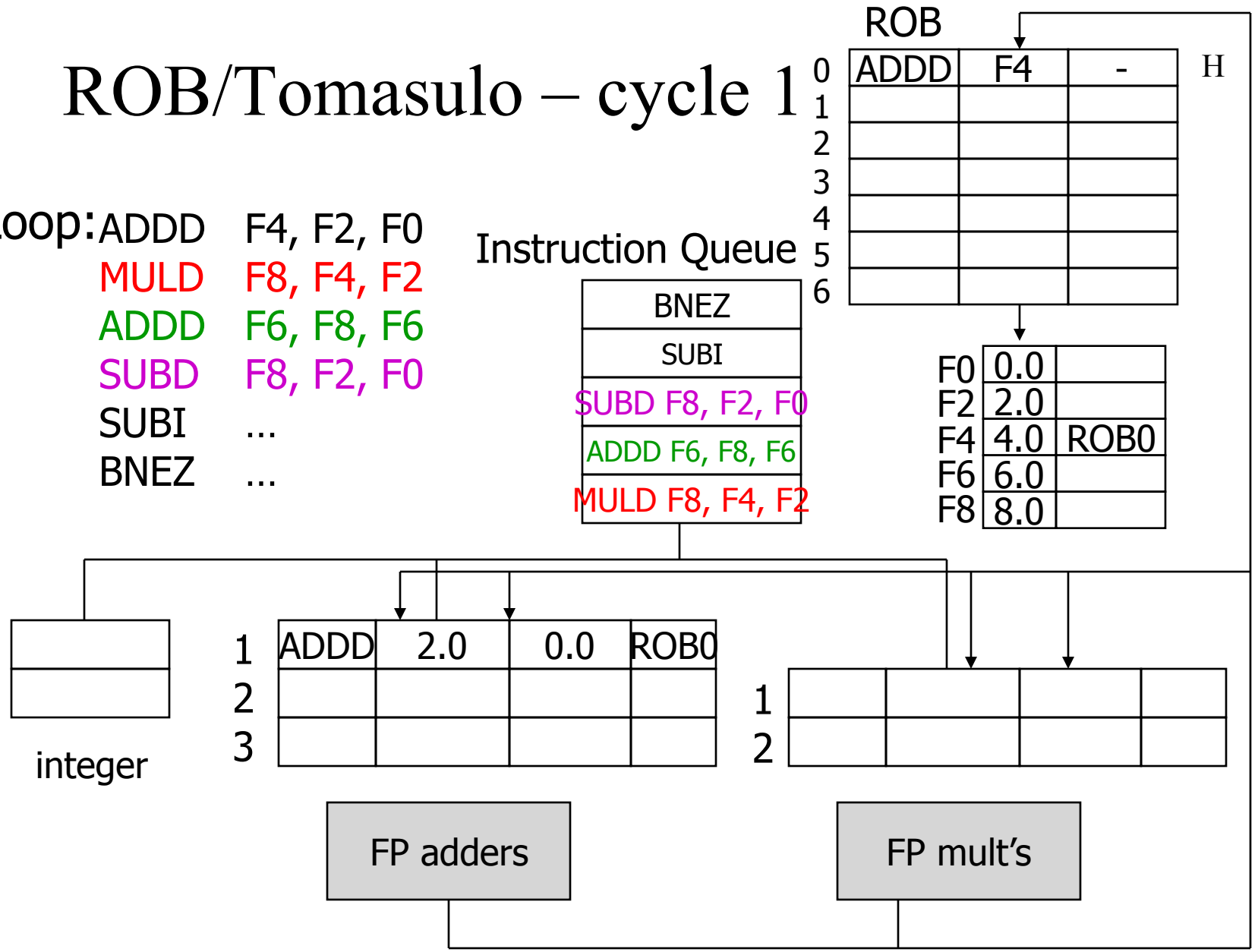
ROB/Tomasulo – cycle 0

```
Loop: ADDD   F4, F2, F0
      MULD   F8, F4, F2
      ADDD   F6, F8, F6
      SUBD   F8, F2, F0
      SUBI   ...
      BNEZ   ..., Loop
```



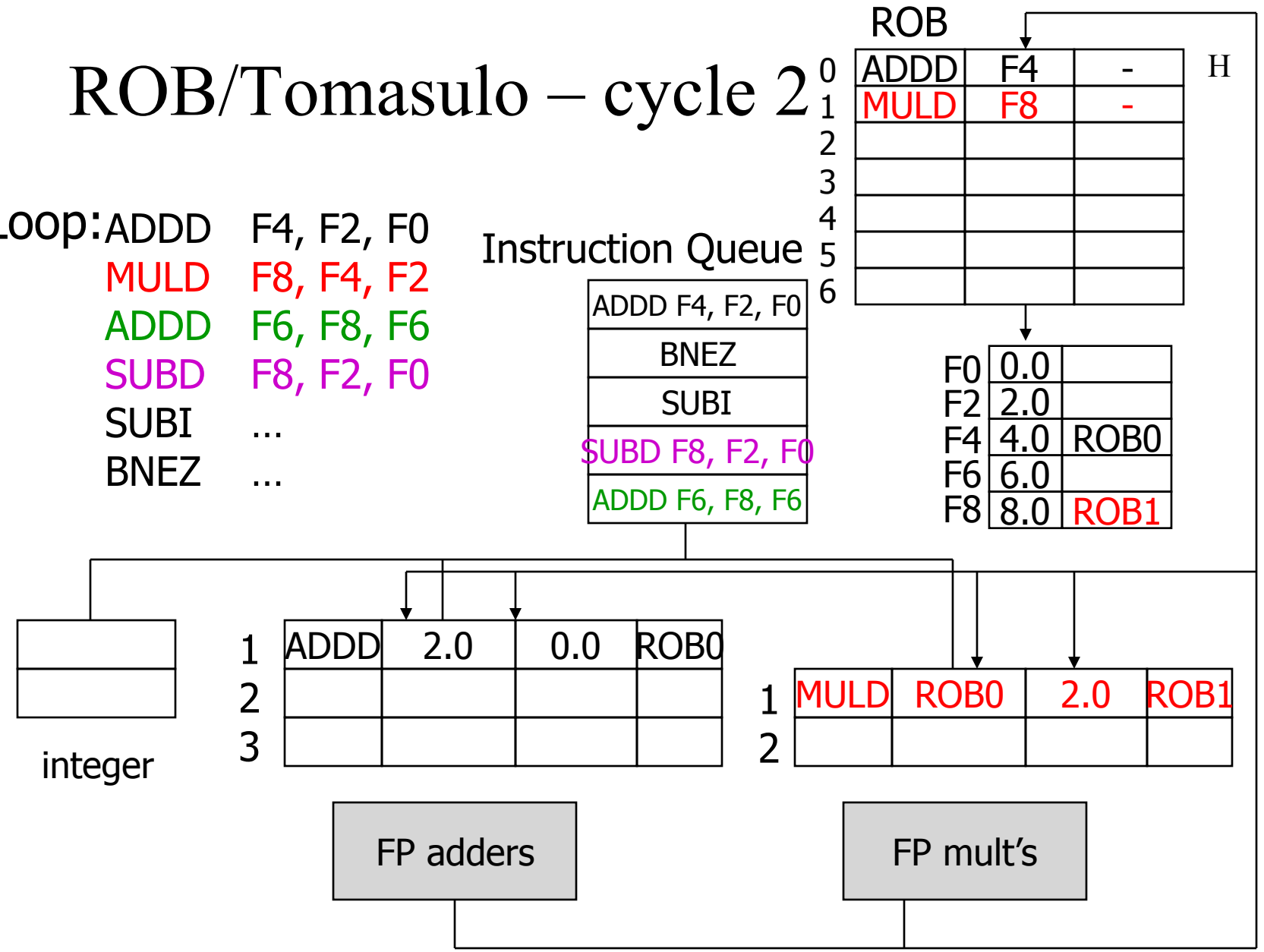
ROB/Tomasulo – cycle 1

```
Loop: ADDD  F4, F2, F0
      MULD  F8, F4, F2
      ADDD  F6, F8, F6
      SUBD  F8, F2, F0
      SUBI  ...
      BNEZ  ...
```



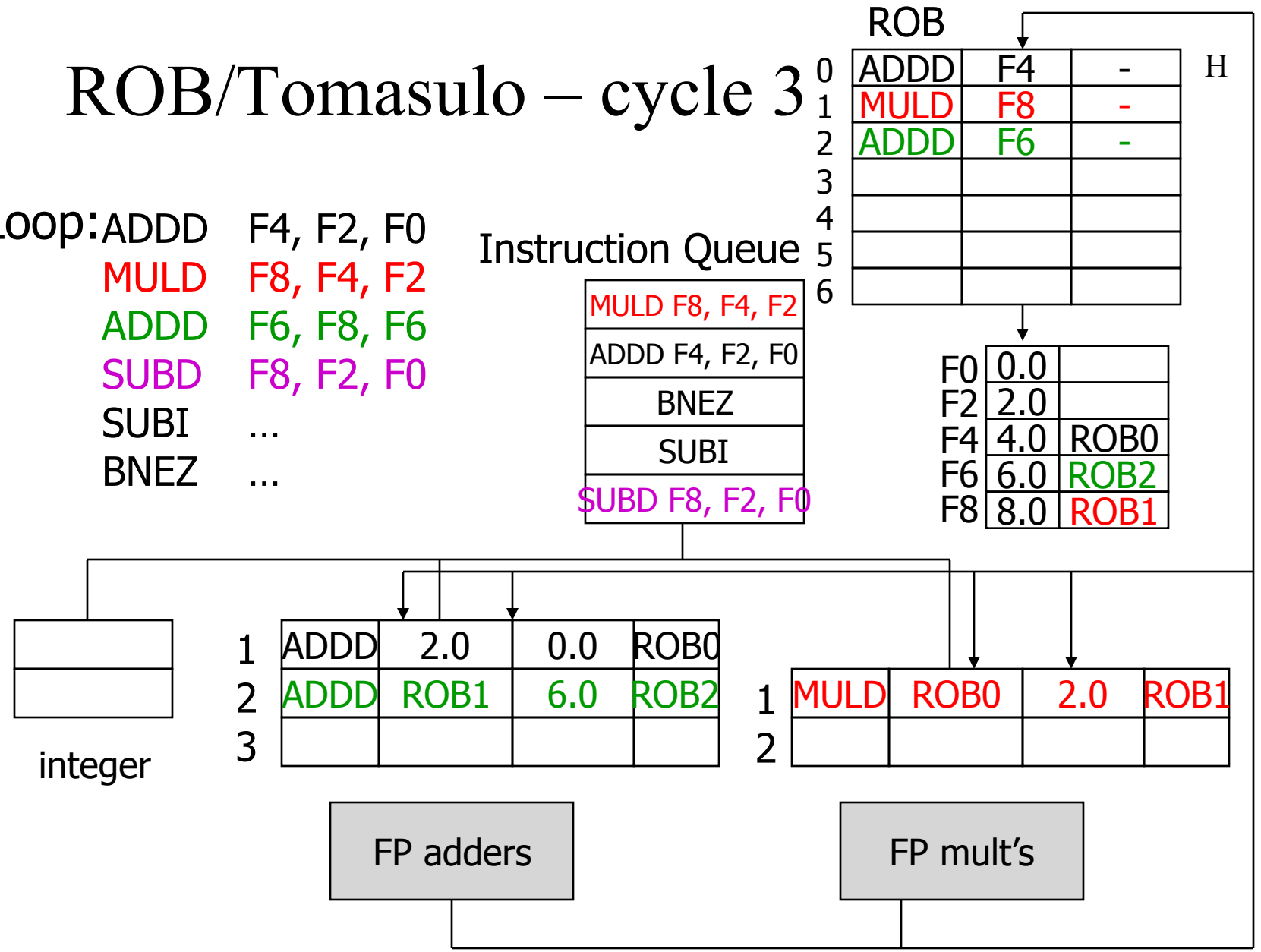
ROB/Tomasulo – cycle 2

```
Loop: ADDD  F4, F2, F0
      MULD  F8, F4, F2
      ADDD  F6, F8, F6
      SUBD  F8, F2, F0
      SUBI  ...
      BNEZ  ...
```



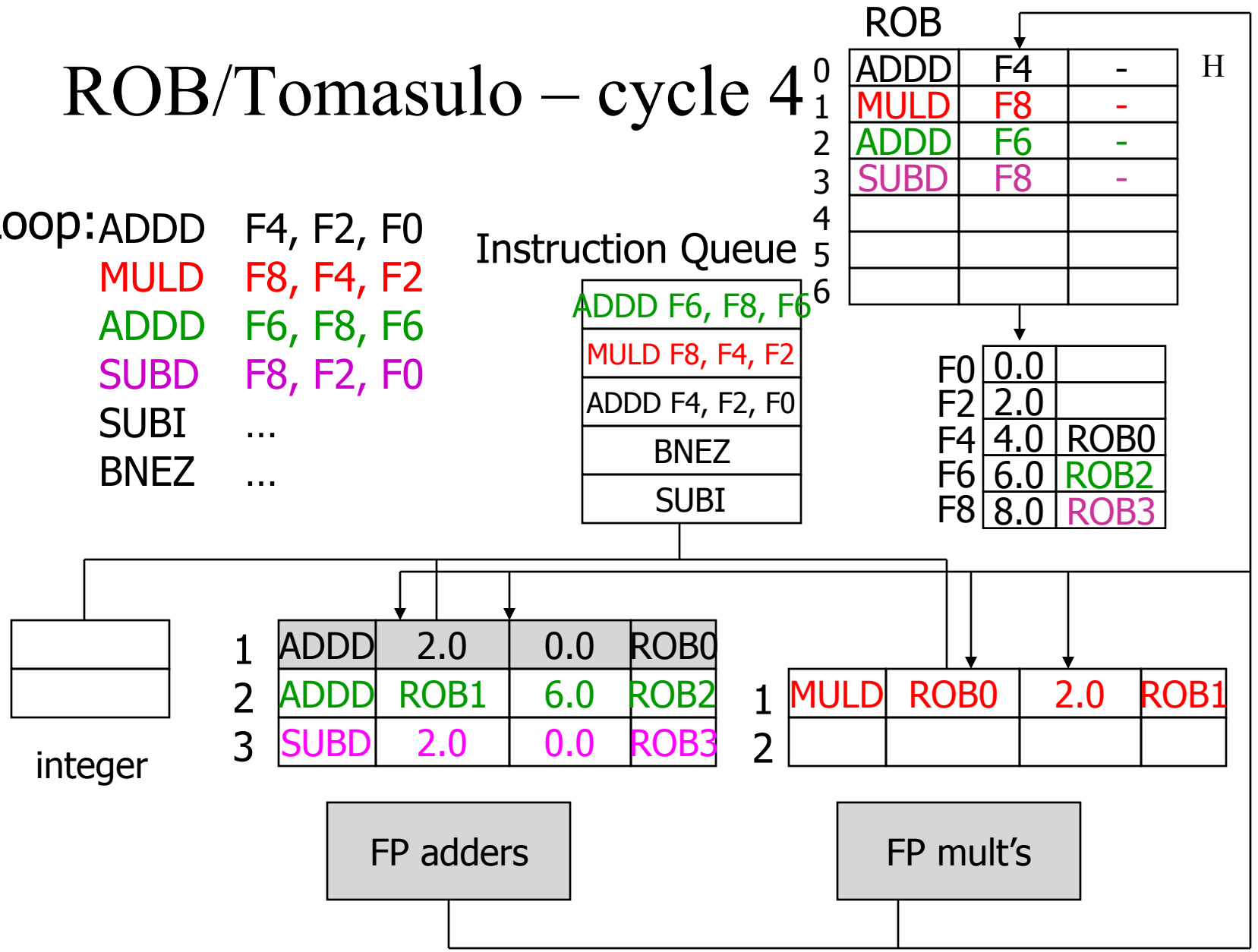
ROB/Tomasulo – cycle 3

```
Loop: ADDD  F4, F2, F0
      MULD  F8, F4, F2
      ADDD  F6, F8, F6
      SUBD  F8, F2, F0
      SUBI  ...
      BNEZ  ...
```



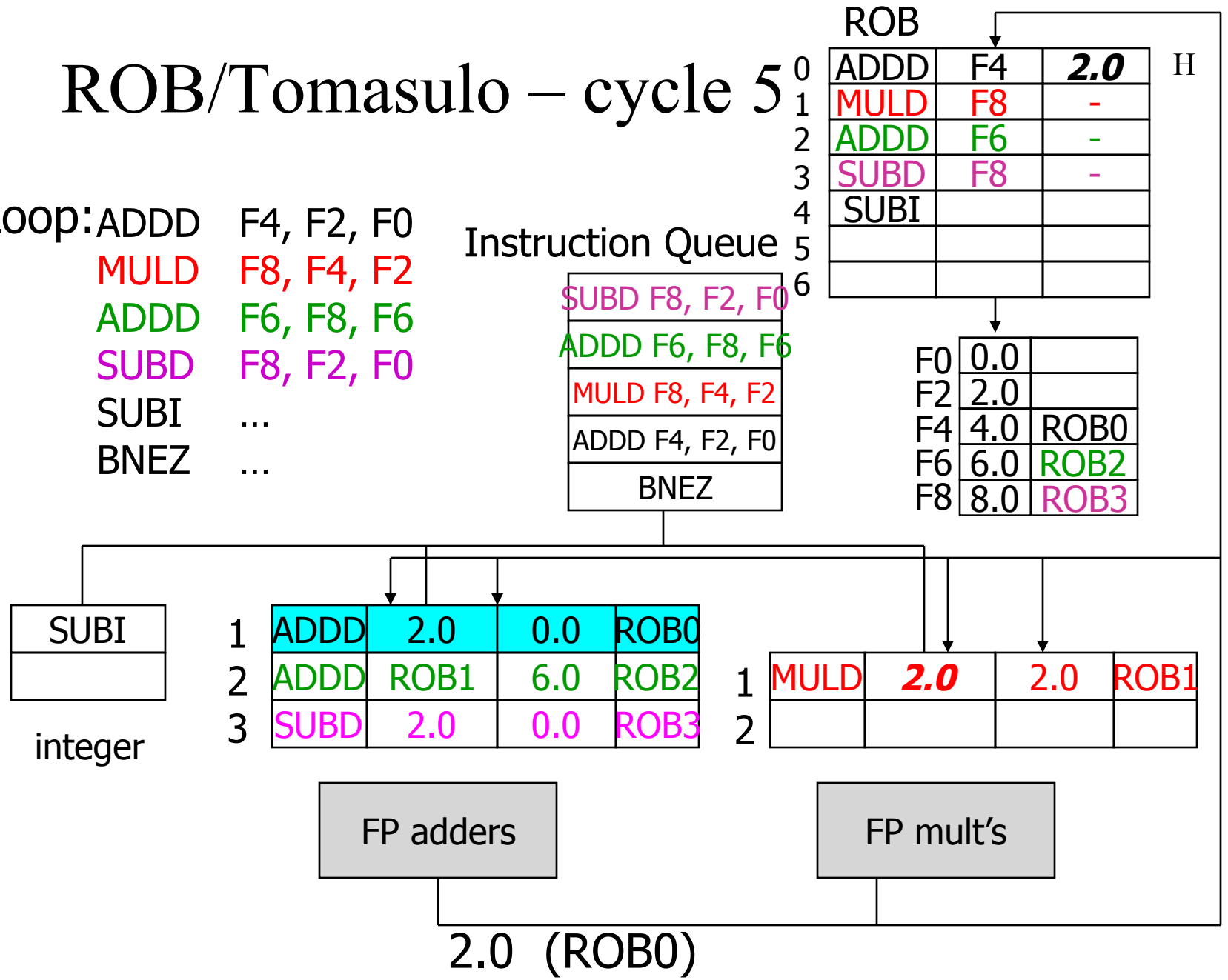
ROB/Tomasulo – cycle 4

```
Loop: ADDD  F4, F2, F0
      MULD  F8, F4, F2
      ADDD  F6, F8, F6
      SUBD  F8, F2, F0
      SUBI  ...
      BNEZ  ...
```



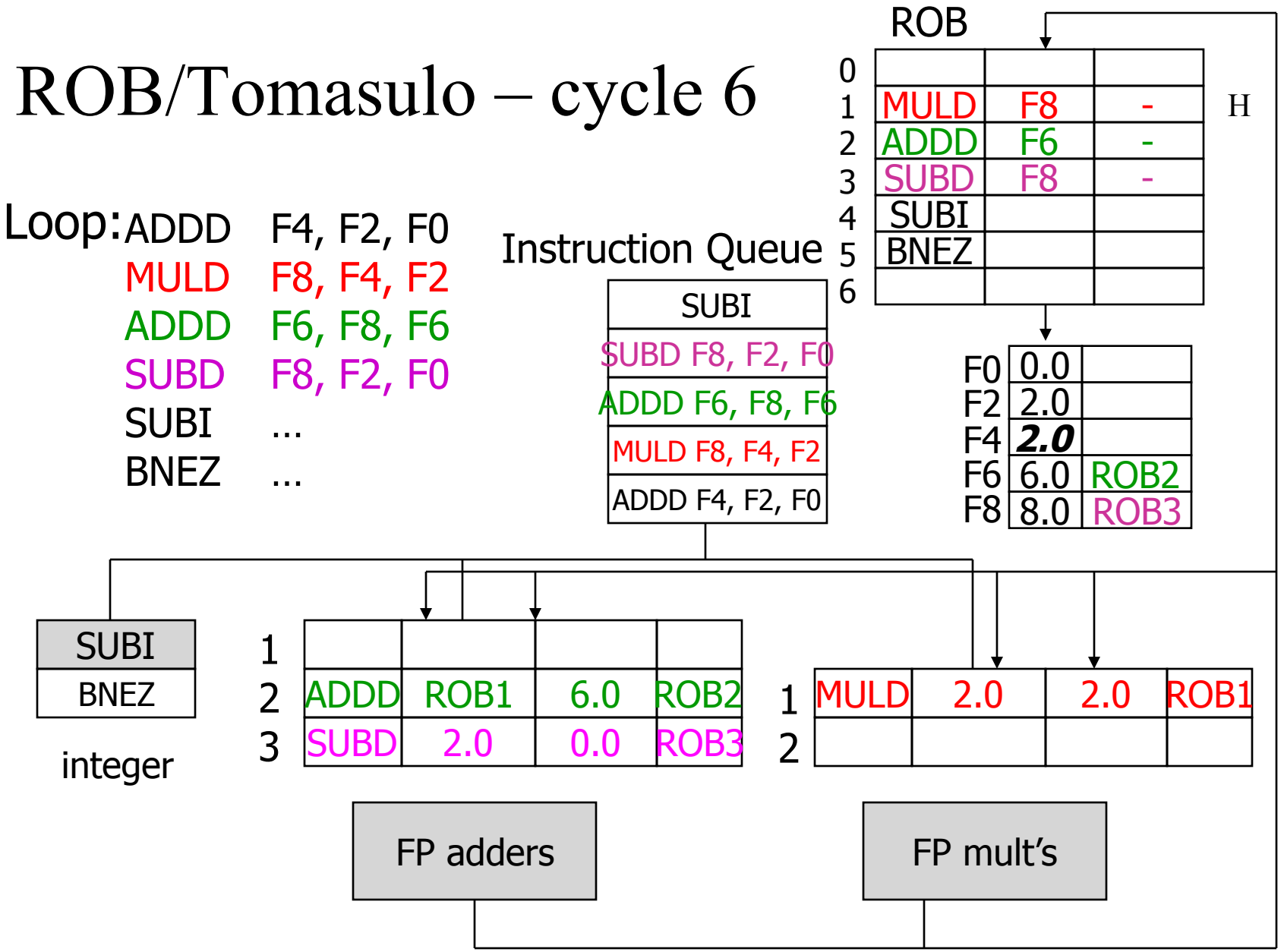
ROB/Tomasulo – cycle 5

```
Loop: ADDD  F4, F2, F0
      MULD  F8, F4, F2
      ADDD  F6, F8, F6
      SUBD  F8, F2, F0
      SUBI  ...
      BNEZ  ...
```



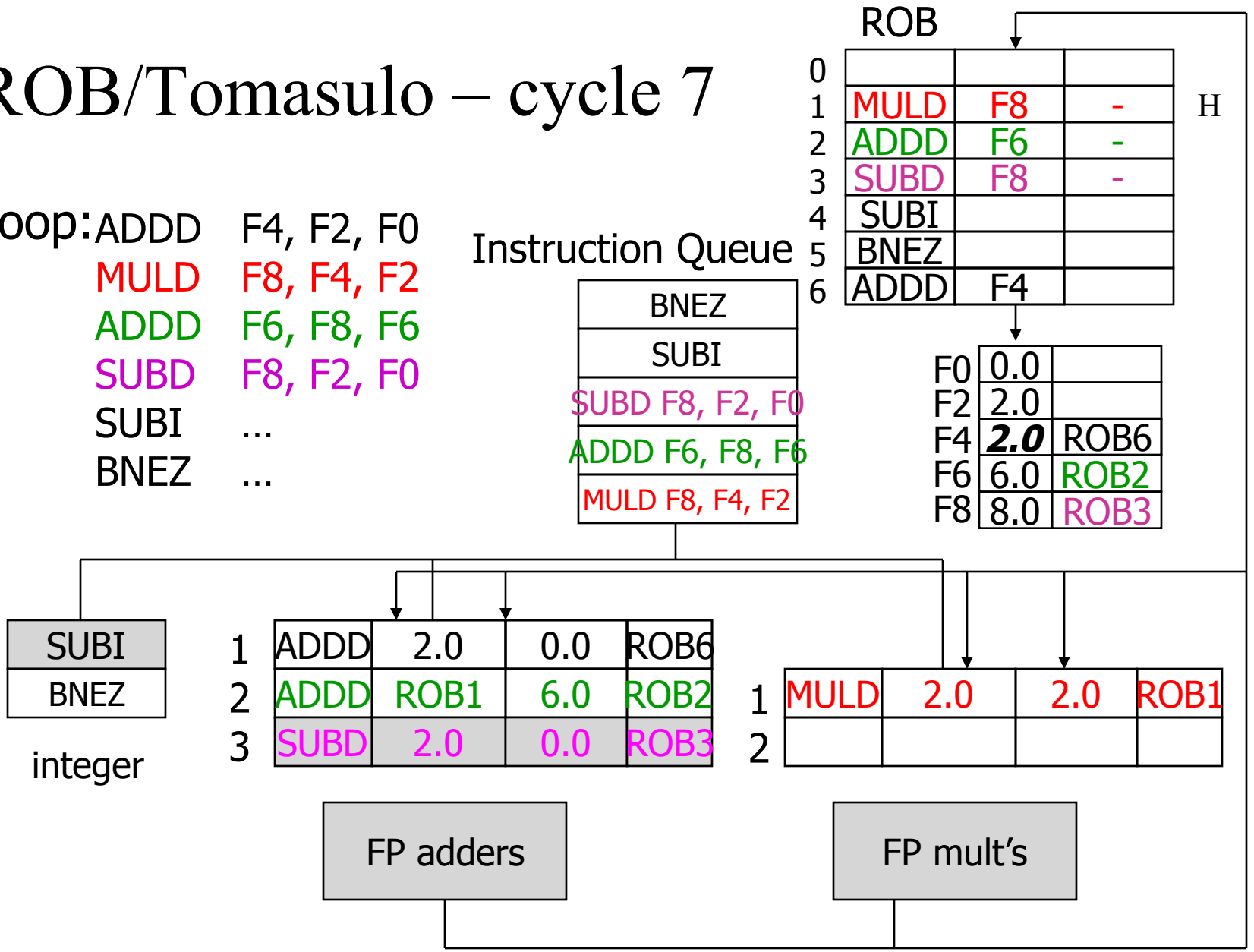
ROB/Tomasulo – cycle 6

```
Loop: ADDD  F4, F2, F0
      MULD  F8, F4, F2
      ADDD  F6, F8, F6
      SUBD  F8, F2, F0
      SUBI  ...
      BNEZ  ...
```

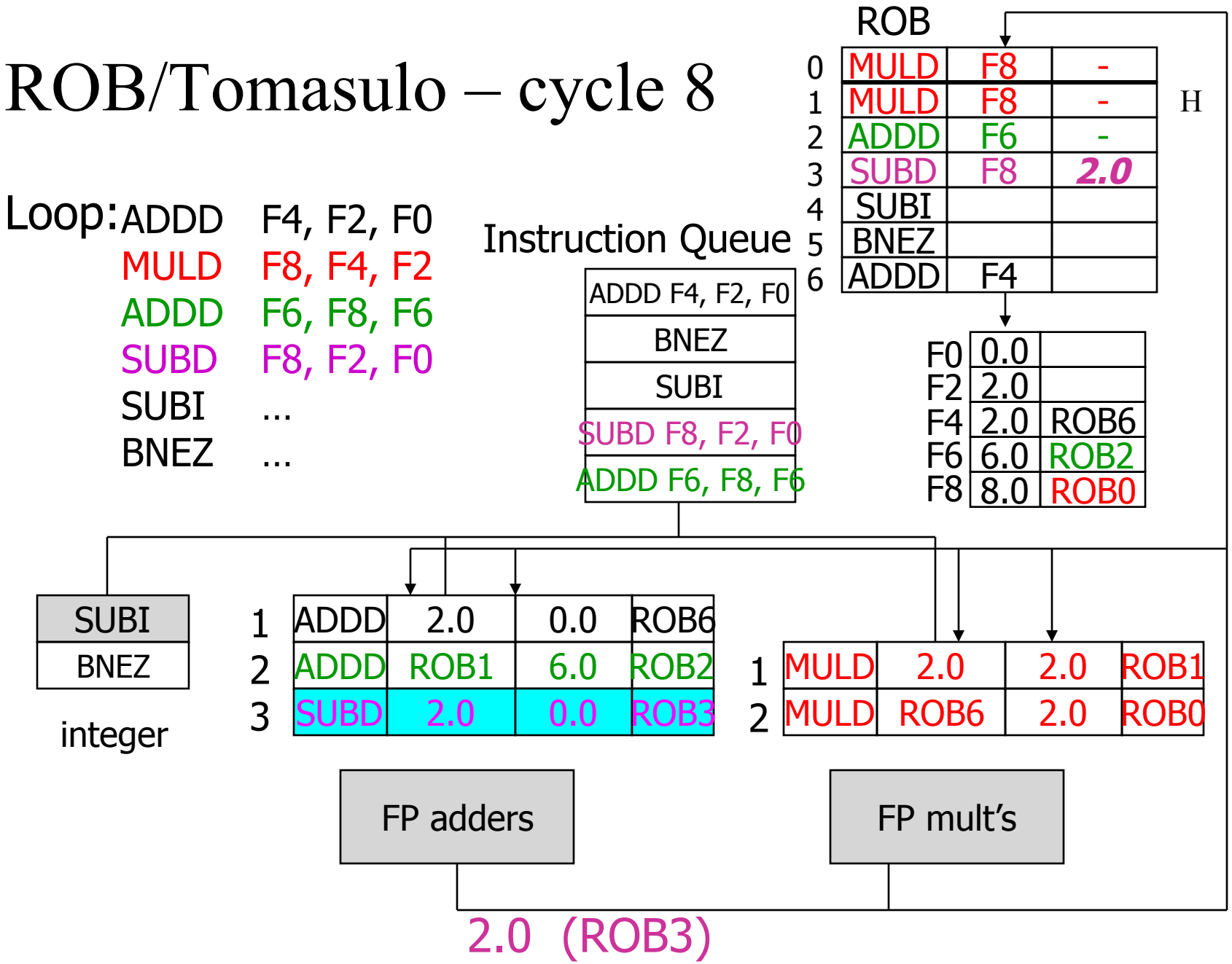


ROB/Tomasulo – cycle 7

```
Loop: ADDD  F4, F2, F0
      MULD  F8, F4, F2
      ADDD  F6, F8, F6
      SUBD  F8, F2, F0
      SUBI  ...
      BNEZ  ...
```

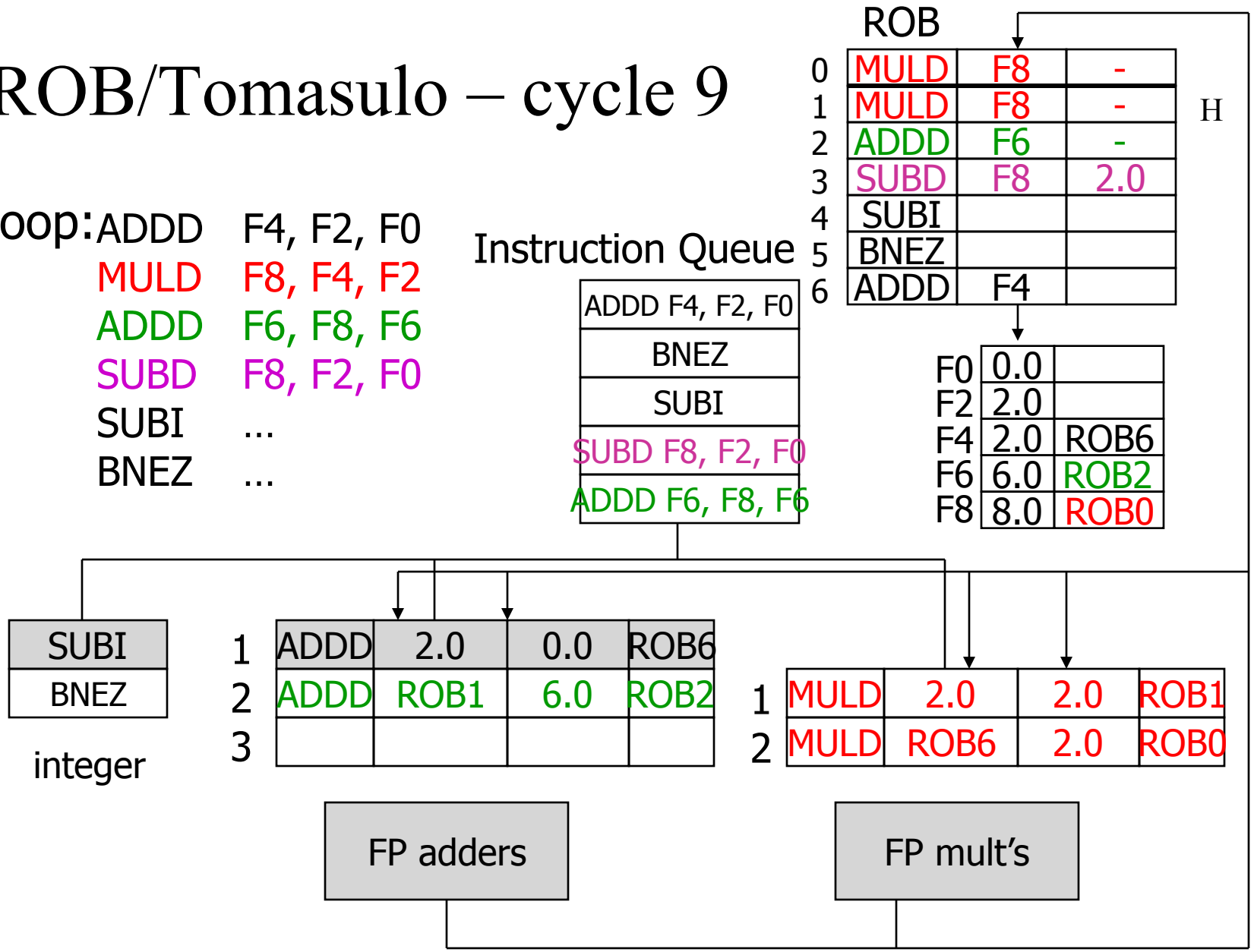


ROB/Tomasulo – cycle 8

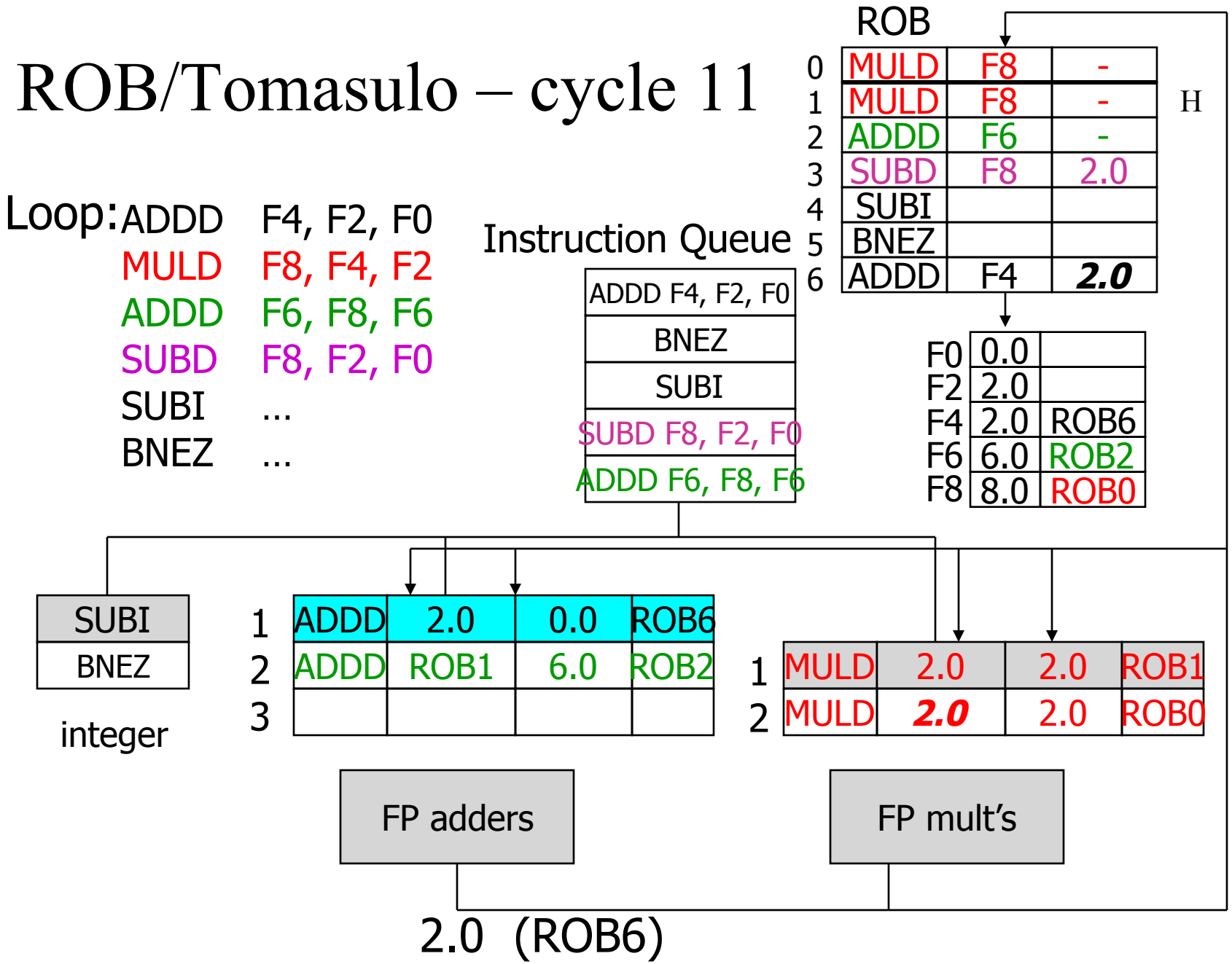


ROB/Tomasulo – cycle 9

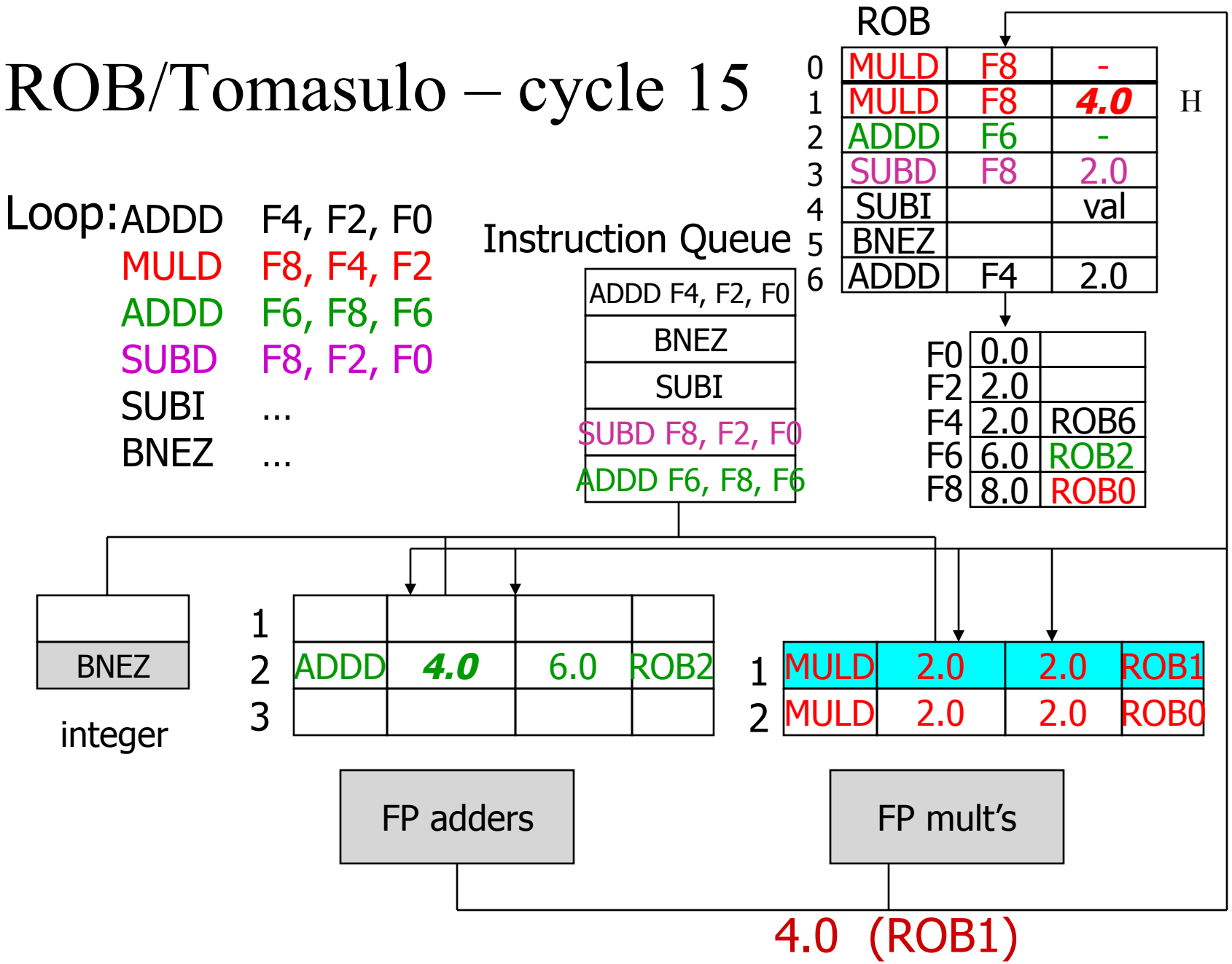
```
Loop: ADDD  F4, F2, F0
      MULD  F8, F4, F2
      ADDD  F6, F8, F6
      SUBD  F8, F2, F0
      SUBI  ...
      BNEZ  ...
```



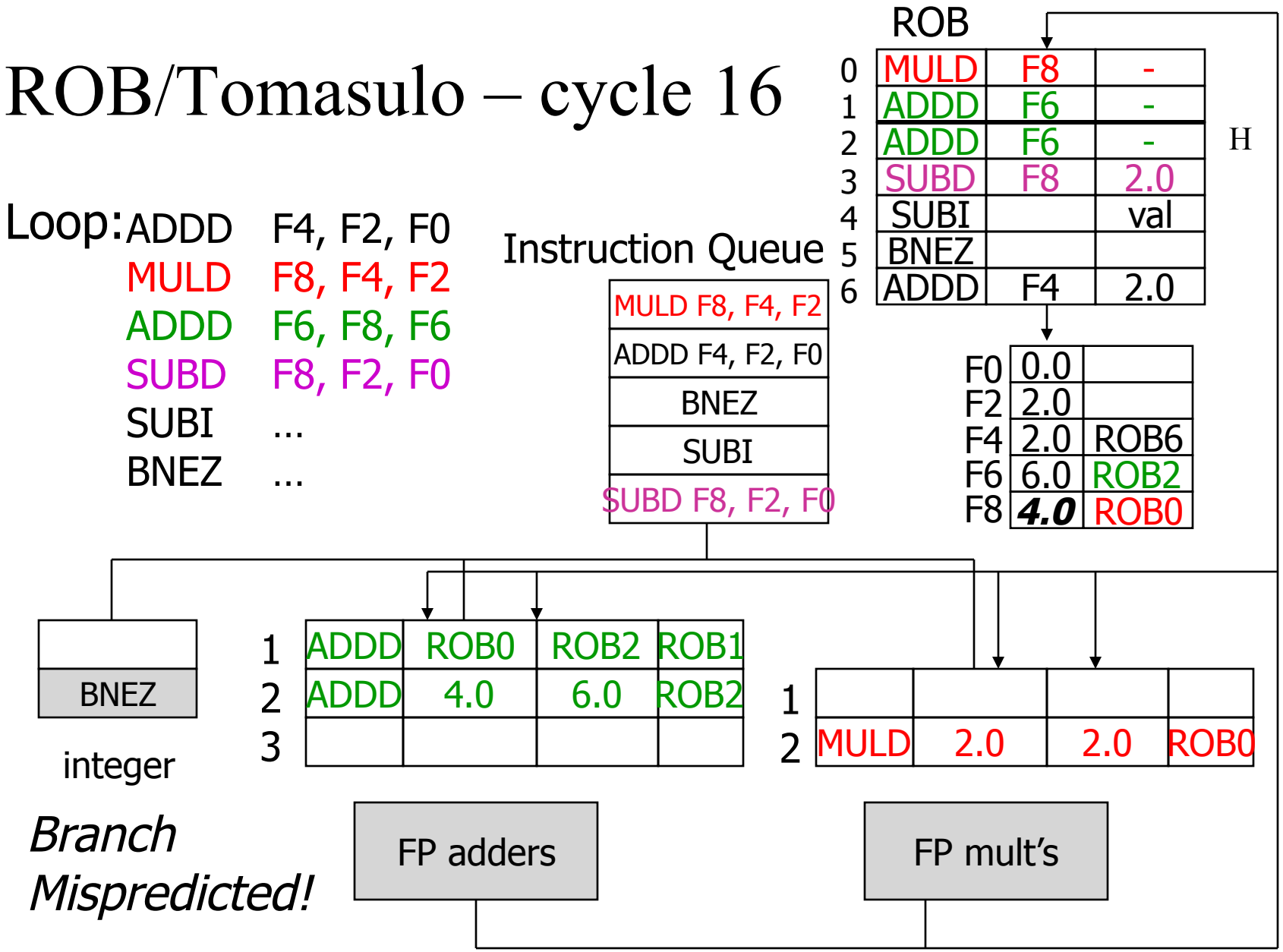
ROB/Tomasulo – cycle 11



ROB/Tomasulo – cycle 15

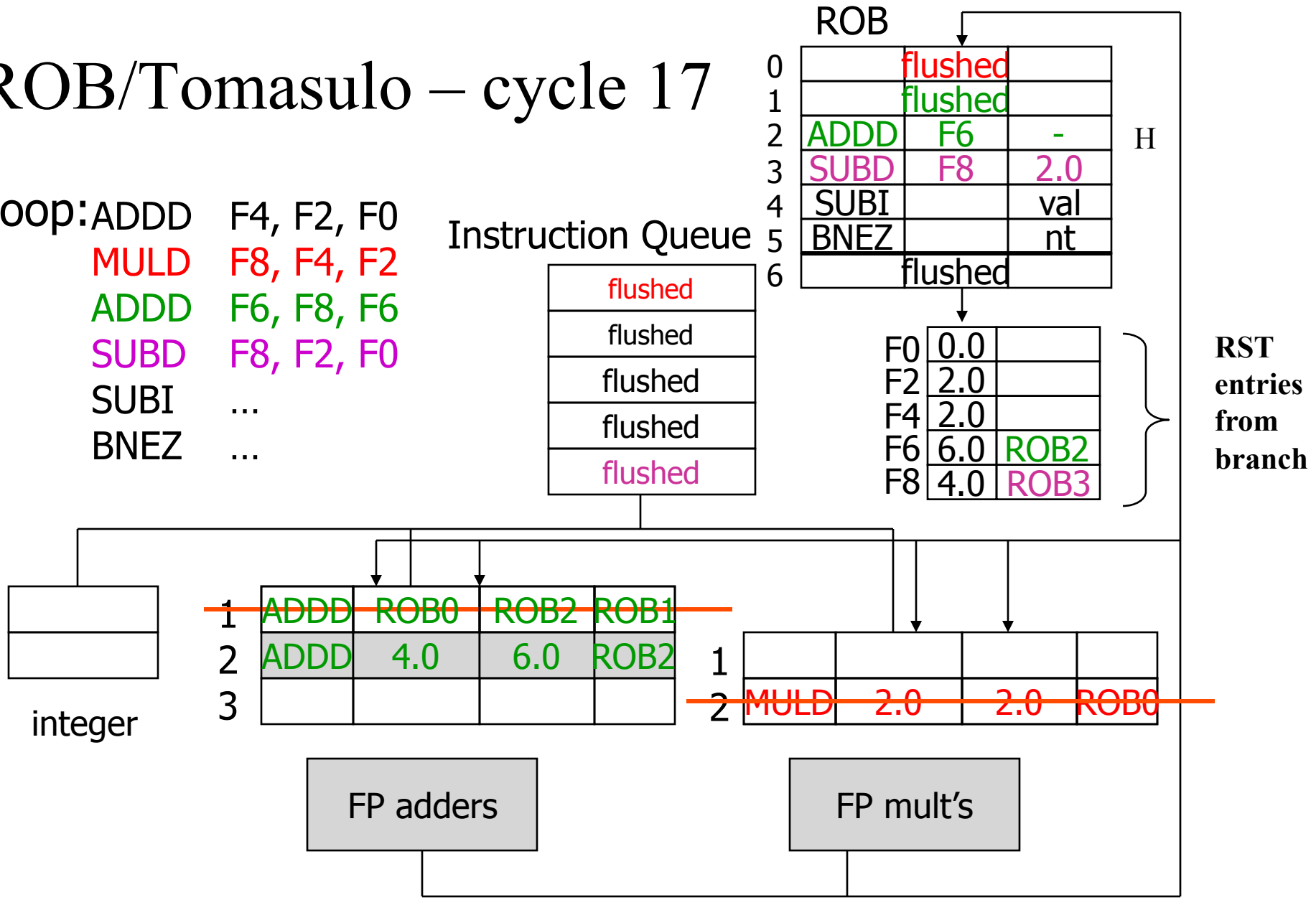


ROB/Tomasulo – cycle 16

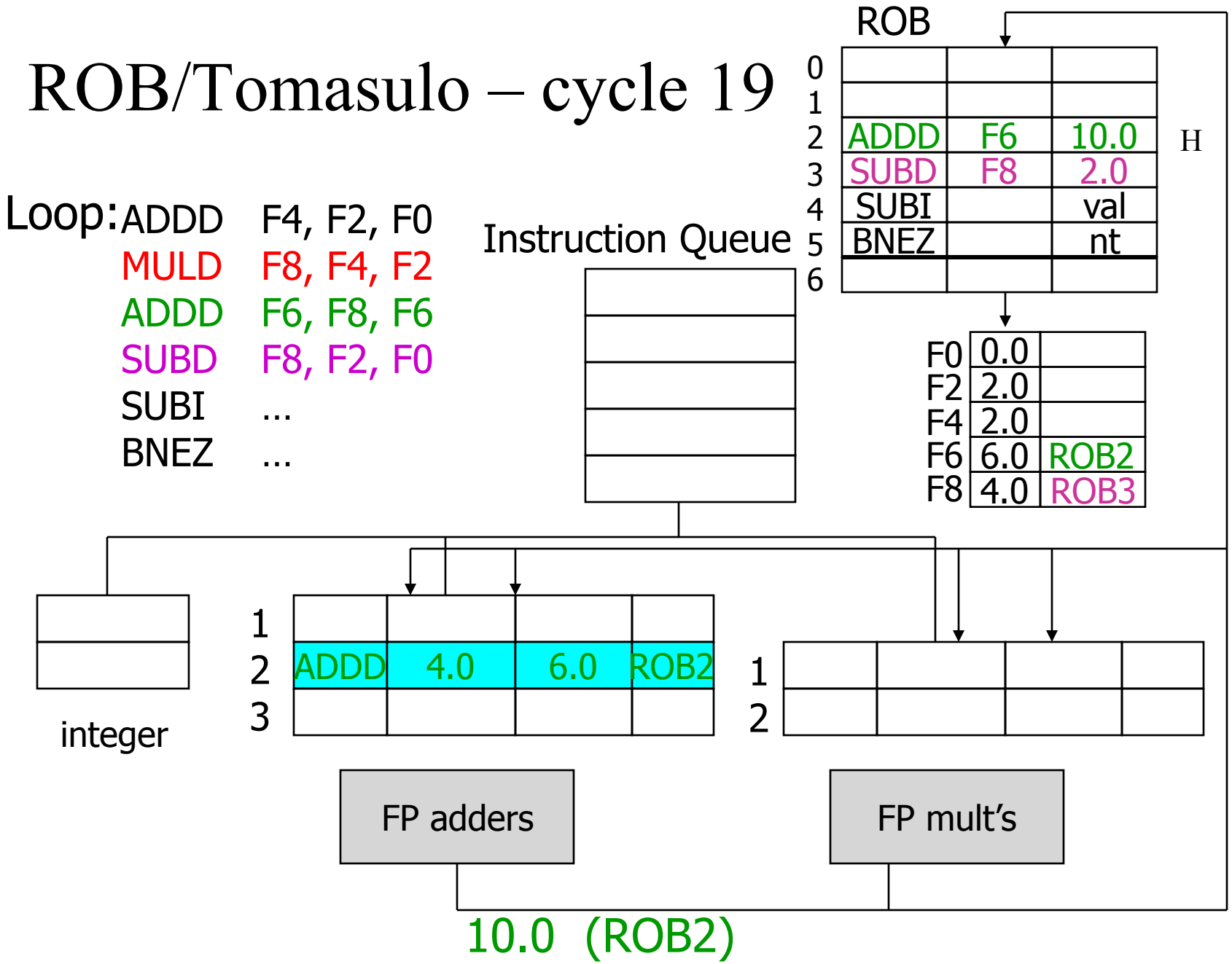


ROB/Tomasulo – cycle 17

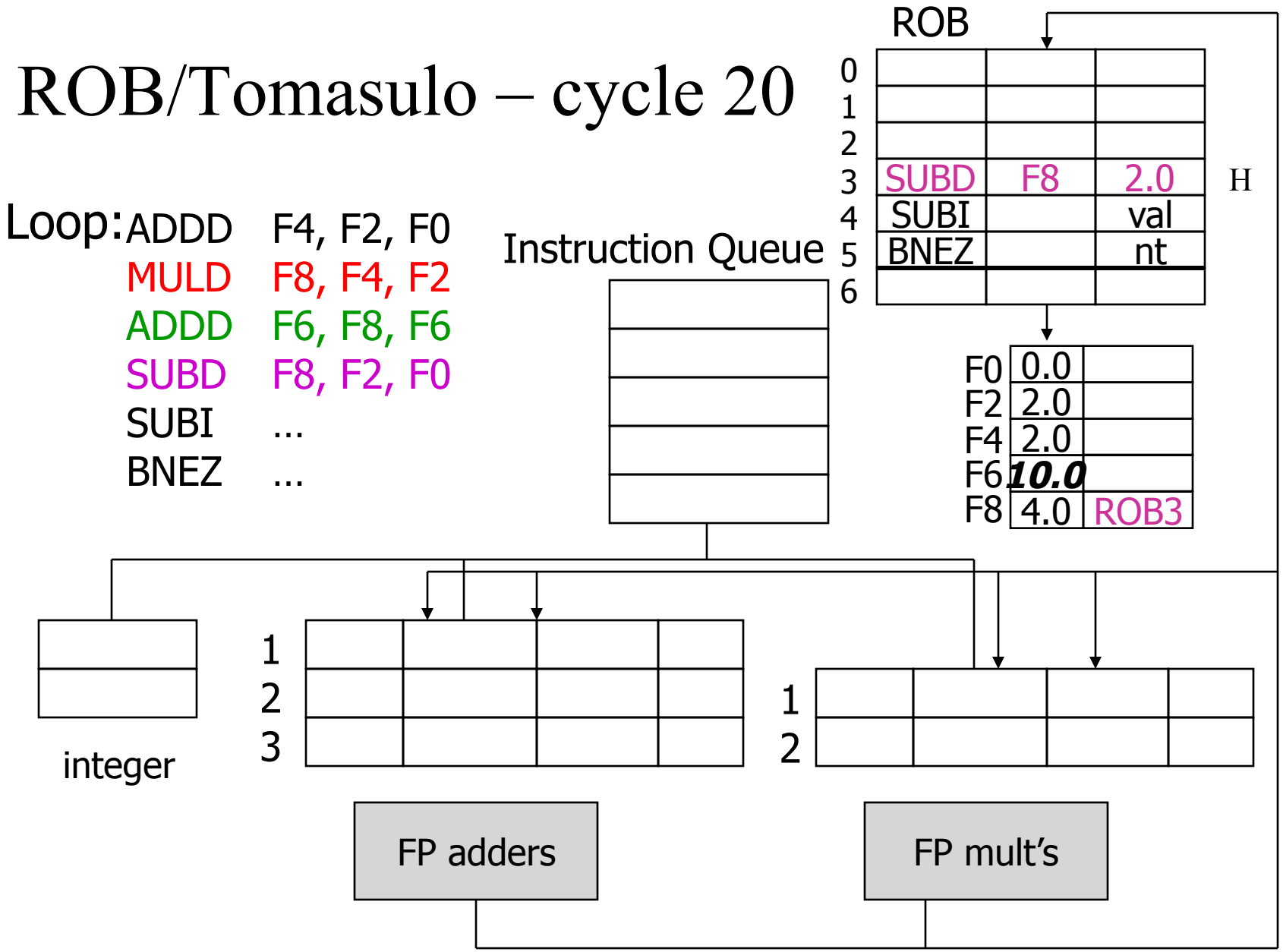
```
Loop: ADDD   F4, F2, F0
      MULD   F8, F4, F2
      ADDD   F6, F8, F6
      SUBD   F8, F2, F0
      SUBI   ...
      BNEZ   ...
```



ROB/Tomasulo – cycle 19

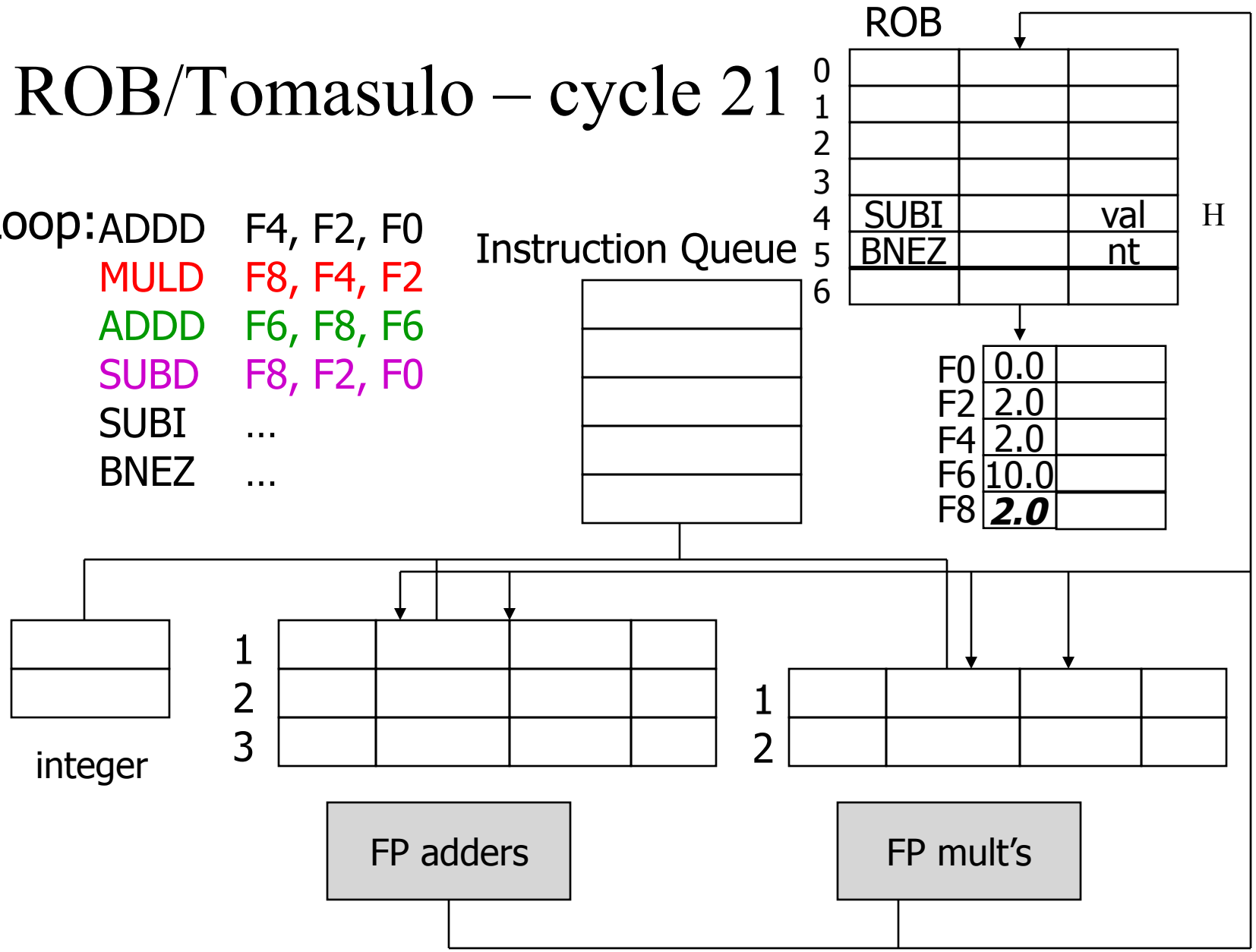


ROB/Tomasulo – cycle 20



ROB/Tomasulo – cycle 21

```
Loop: ADDD  F4, F2, F0
      MULD  F8, F4, F2
      ADDD  F6, F8, F6
      SUBD  F8, F2, F0
      SUBI  ...
      BNEZ  ...
```



HW support for More ILP

- *Speculation*: allow an instruction to issue that is dependent on branch predicted to be taken *without* any consequences (including exceptions) if branch is not actually taken ("HW undo")
- Often combined with dynamic scheduling
- Tomasulo: separate *speculative* bypassing of results from real bypassing of results
  - When instruction no longer speculative, write results (*instruction commit*)
  - execute out-of-order but commit in order

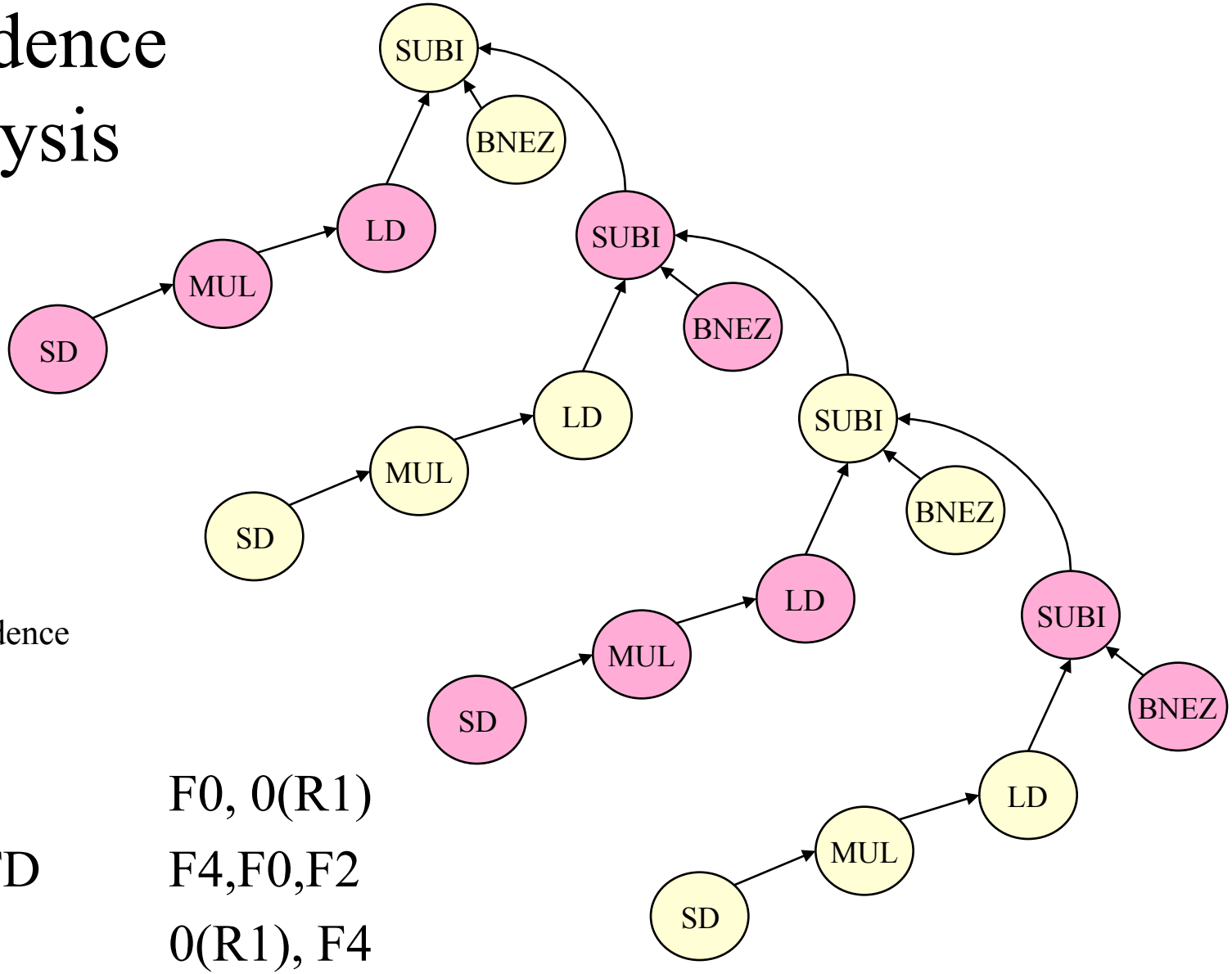
Our Favorite Loop Revisited

```
Loop:  LD      F0, 0(R1)
       MULTD   F4,F0,F2
       SD      0(R1), F4
       SUBI    R1,R1,#8
       BNEZ    R1, Loop
```

Analyze impact of:

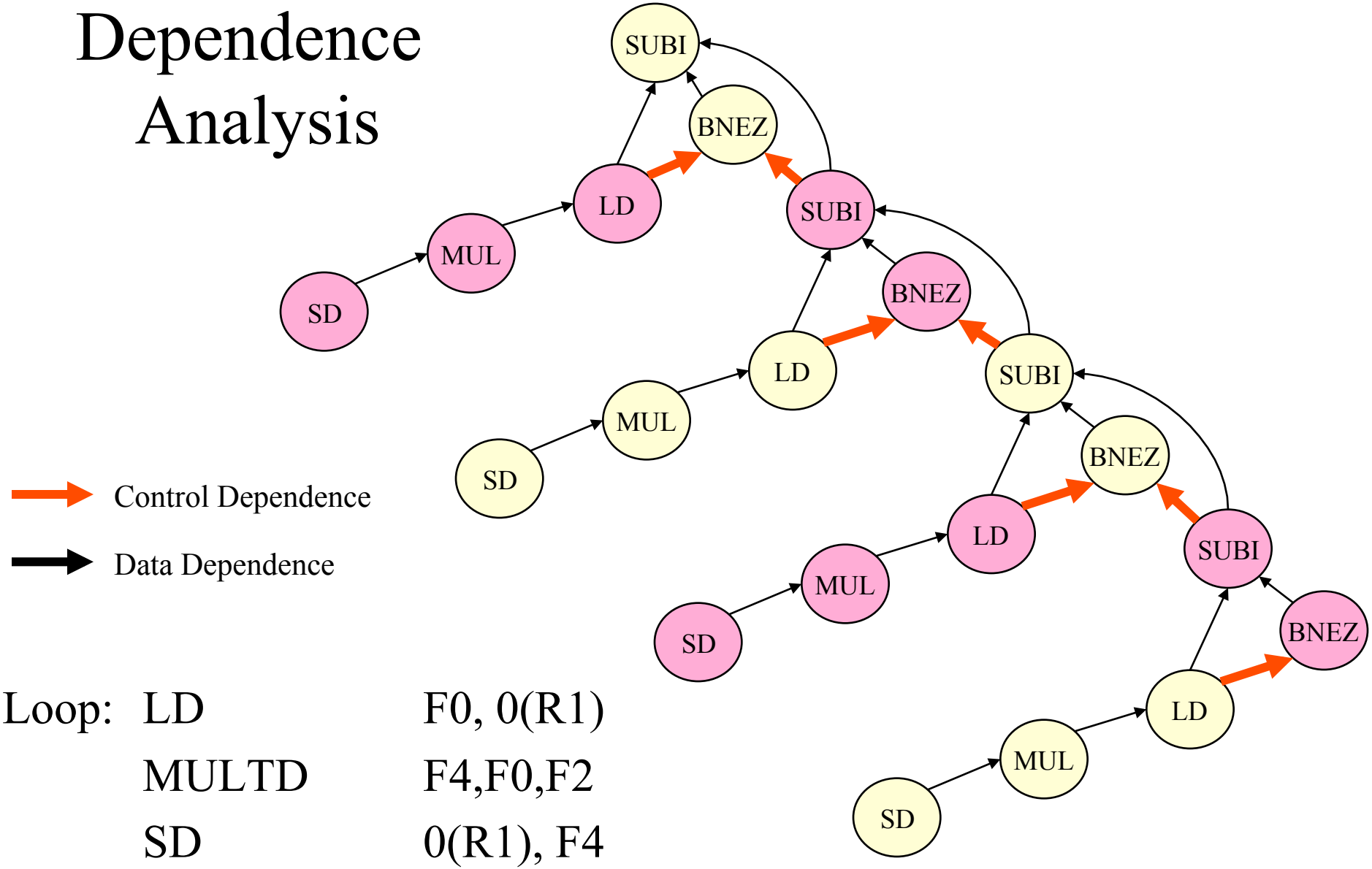
Register Renaming, (360/91)
Memory Disambiguation, (360/91)
Control Speculation, (ROB, etc)
and Memory Speculation (Some modern superscalars)

Dependence Analysis



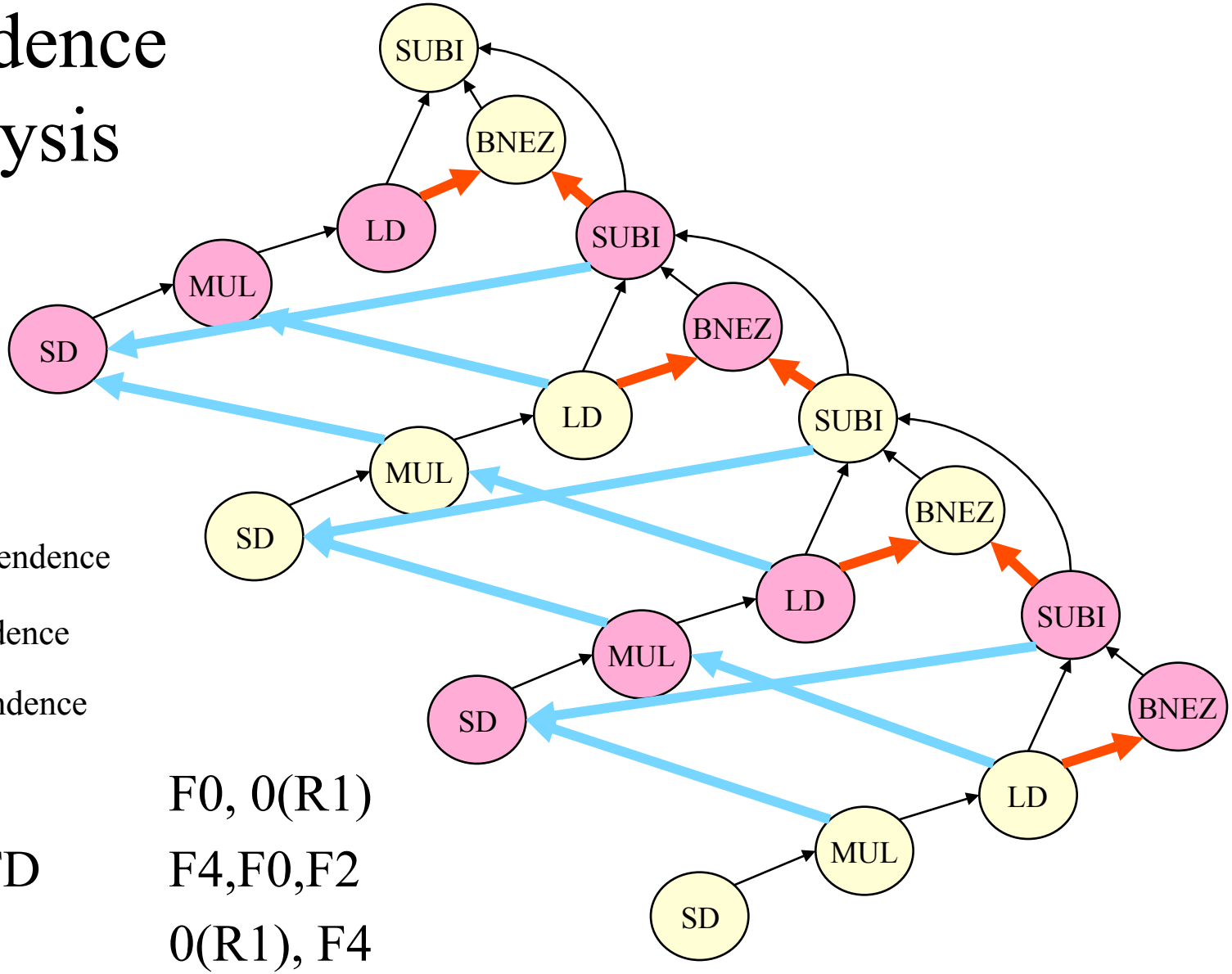

```
Loop:  LD      F0, 0(R1)
       MULTD   F4,F0,F2
       SD      0(R1), F4
       SUBI    R1,R1,#8
       BNEZ    R1, Loop
```

Dependence Analysis




```
Loop:  LD      F0, 0(R1)
       MULTD   F4,F0,F2
       SD      0(R1), F4
       SUBI    R1,R1,#8
       BNEZ    R1, Loop
```

Dependence Analysis



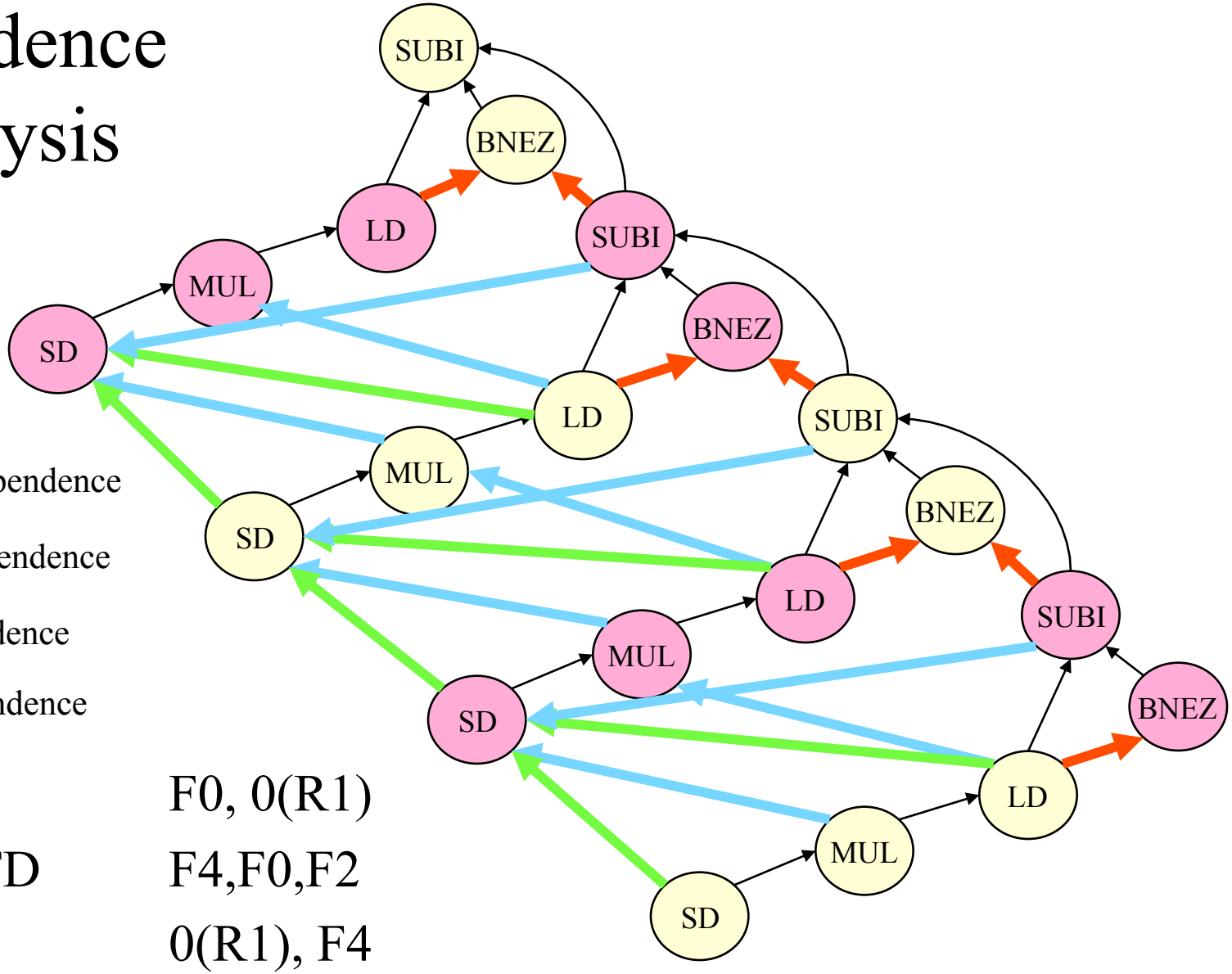
Control Dependence

Data Dependence

Name Dependence

```
Loop:  LD      F0, 0(R1)
       MULTD   F4,F0,F2
       SD      0(R1), F4
       SUBI    R1,R1,#8
       BNEZ    R1, Loop
```

Dependence Analysis



Memory Dependence

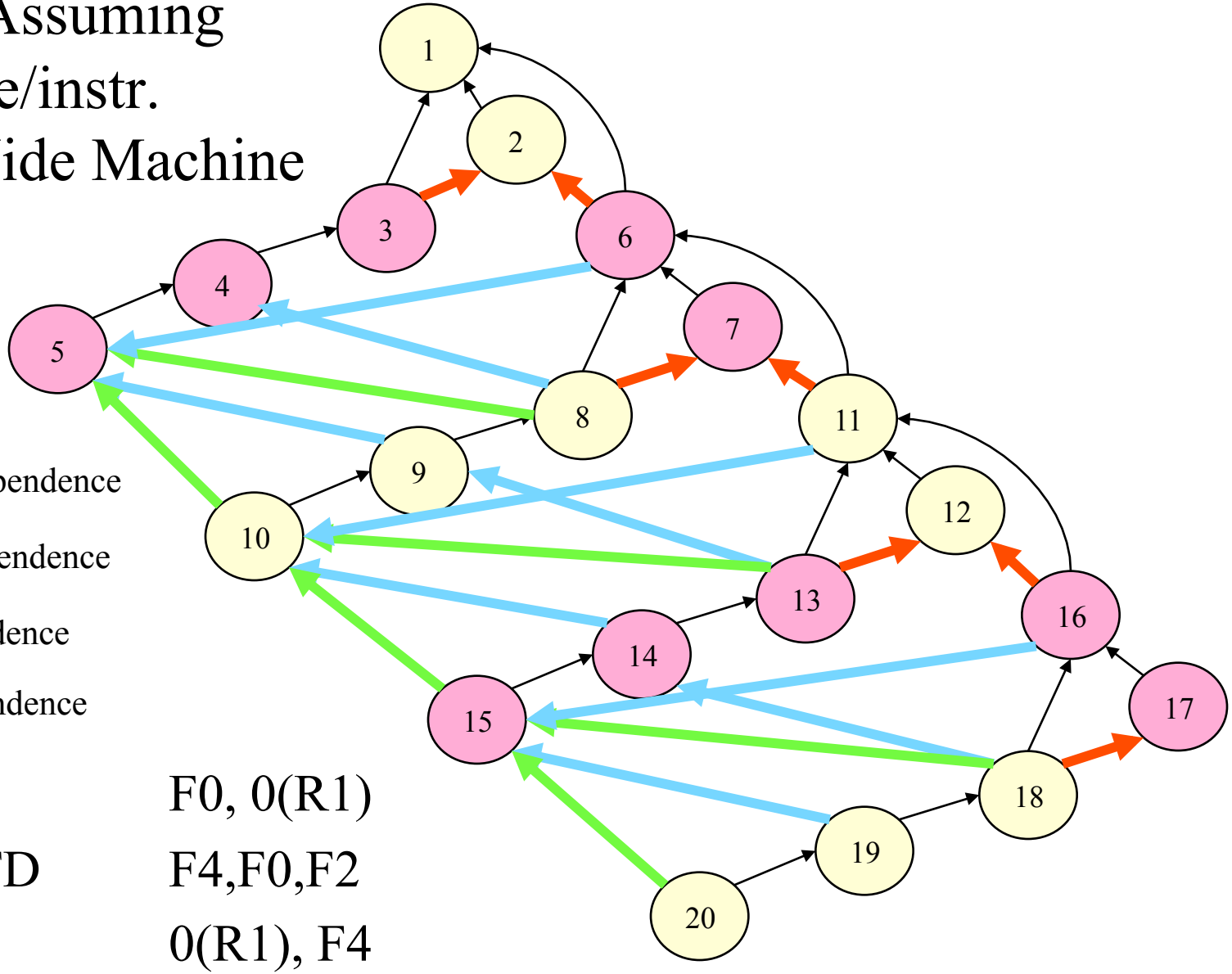
Control Dependence

Data Dependence

Name Dependence

```
Loop:  LD      F0, 0(R1)
       MULTD   F4,F0,F2
       SD      0(R1), F4
       SUBI    R1,R1,#8
       BNEZ    R1, Loop
```

Timing Assuming 1 cycle/instr. Infinitely Wide Machine



Memory Dependence

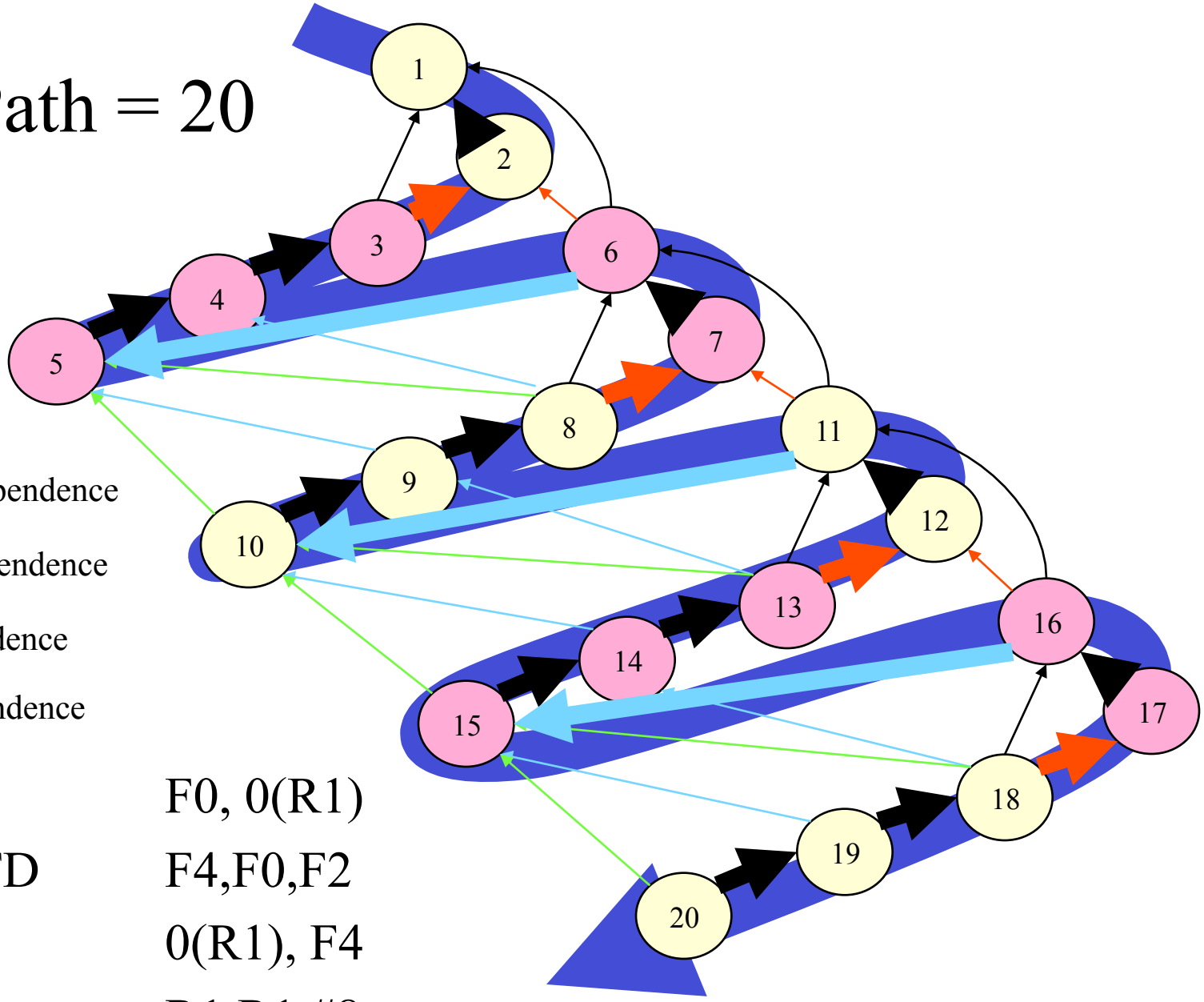
Control Dependence

Data Dependence

Name Dependence

```
Loop:  LD      F0, 0(R1)
       MULTD   F4,F0,F2
       SD      0(R1), F4
       SUBI    R1,R1,#8
       BNEZ    R1, Loop
```

Critical Path = 20



Memory Dependence

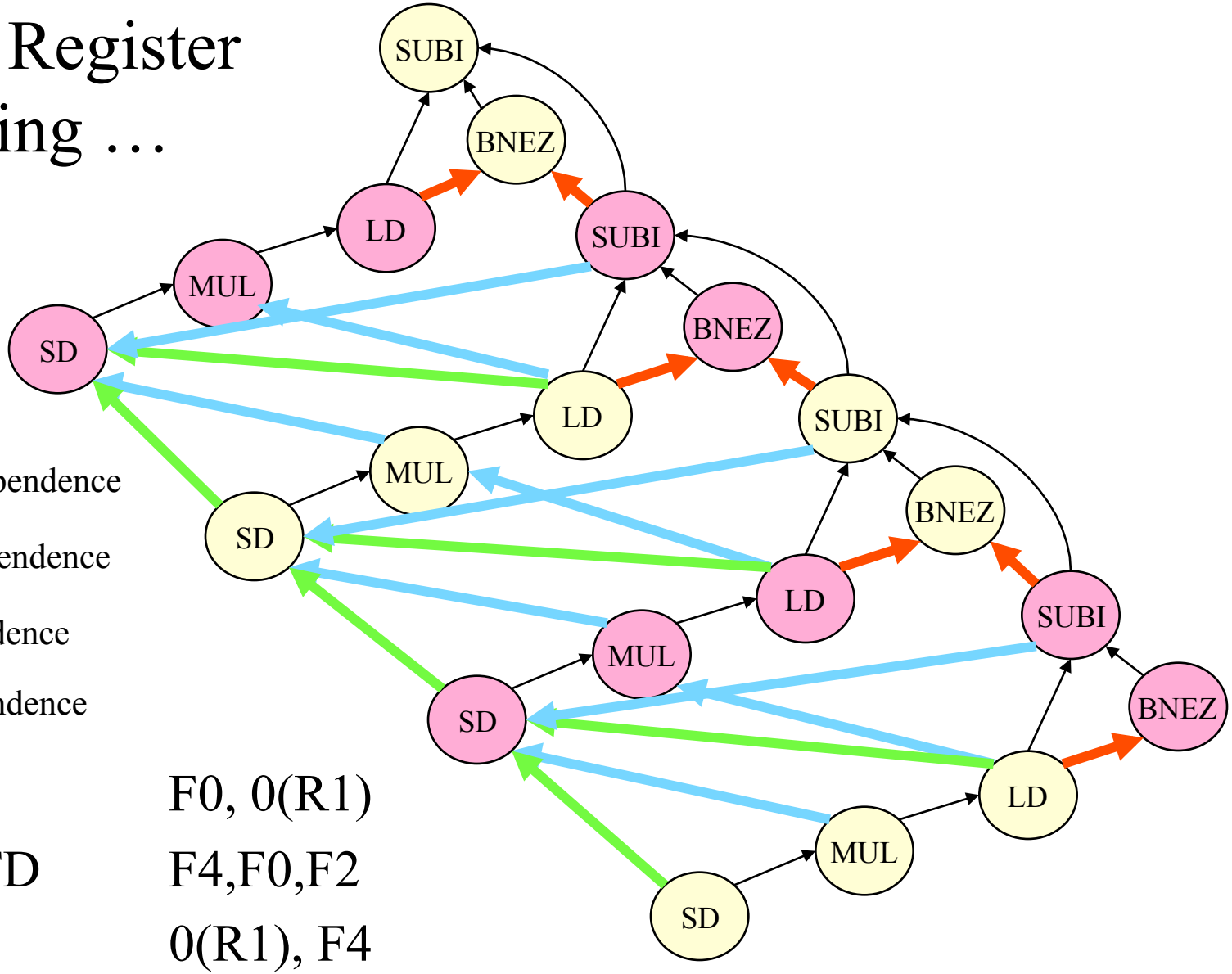
Control Dependence

Data Dependence

Name Dependence

```
Loop:  LD      F0, 0(R1)
       MULTD   F4,F0,F2
       SD      0(R1), F4
       SUBI    R1,R1,#8
       BNEZ    R1, Loop
```

Let's Add Register Renaming …



Memory Dependence

Control Dependence

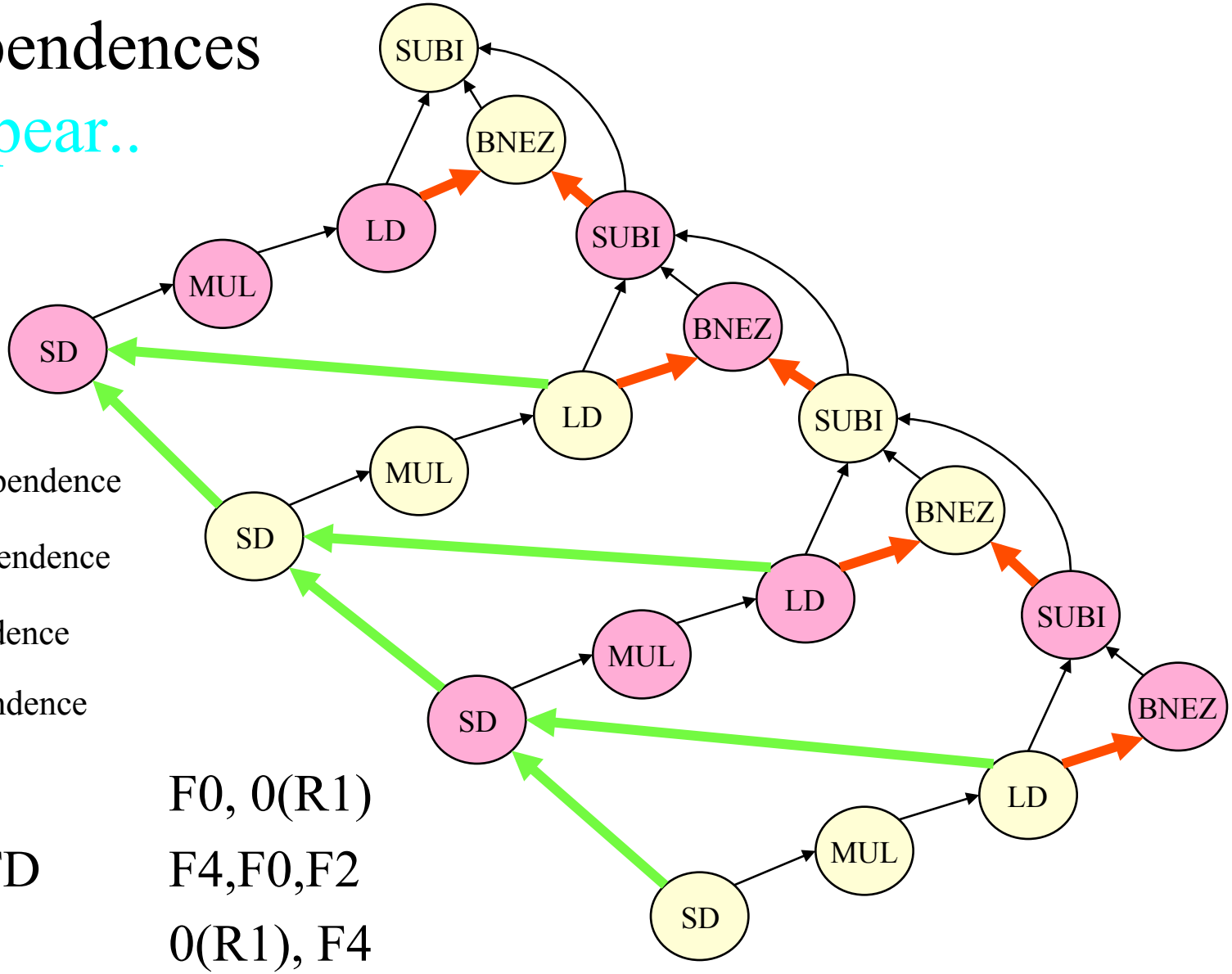
Data Dependence

Name Dependence

```
Loop:  LD      F0, 0(R1)
       MULTD   F4,F0,F2
       SD      0(R1), F4
       SUBI    R1,R1,#8
       BNEZ    R1, Loop
```

Name Dependences

Disappear..



Memory Dependence

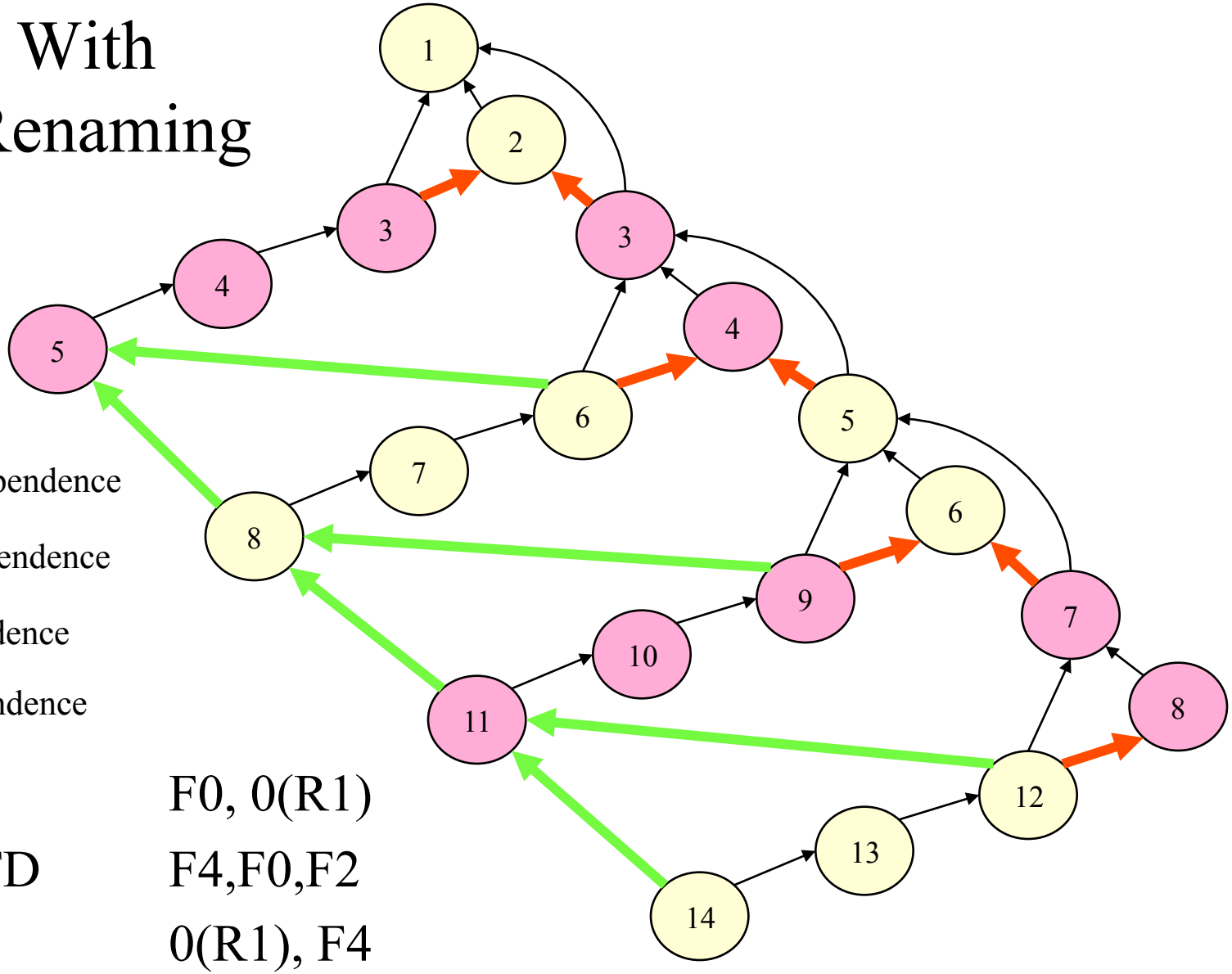
Control Dependence

Data Dependence

Name Dependence

```
Loop:  LD      F0, 0(R1)
       MULTD   F4,F0,F2
       SD      0(R1), F4
       SUBI    R1,R1,#8
       BNEZ    R1, Loop
```

Timing With Register Renaming



Memory Dependence

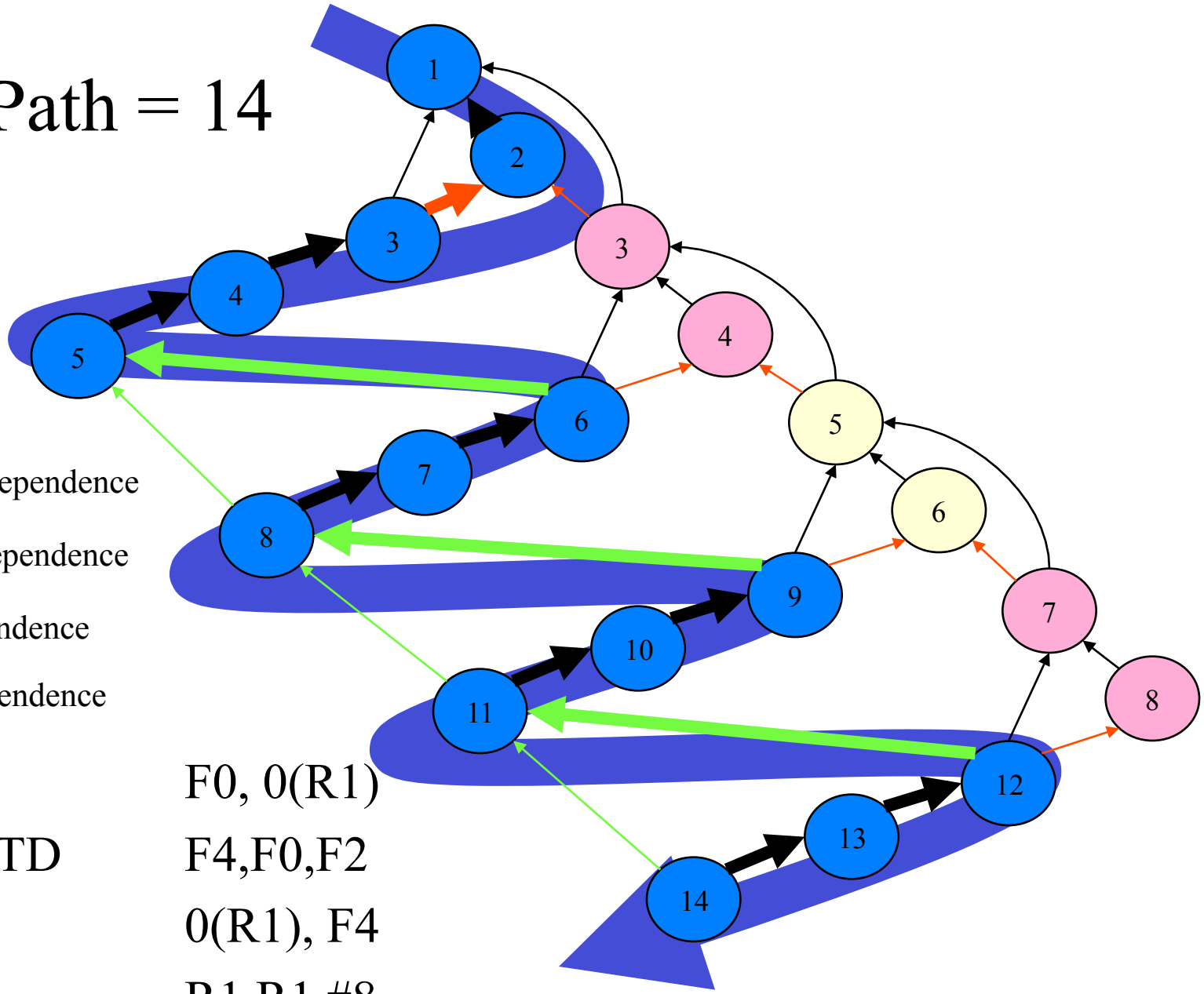
Control Dependence

Data Dependence

Name Dependence

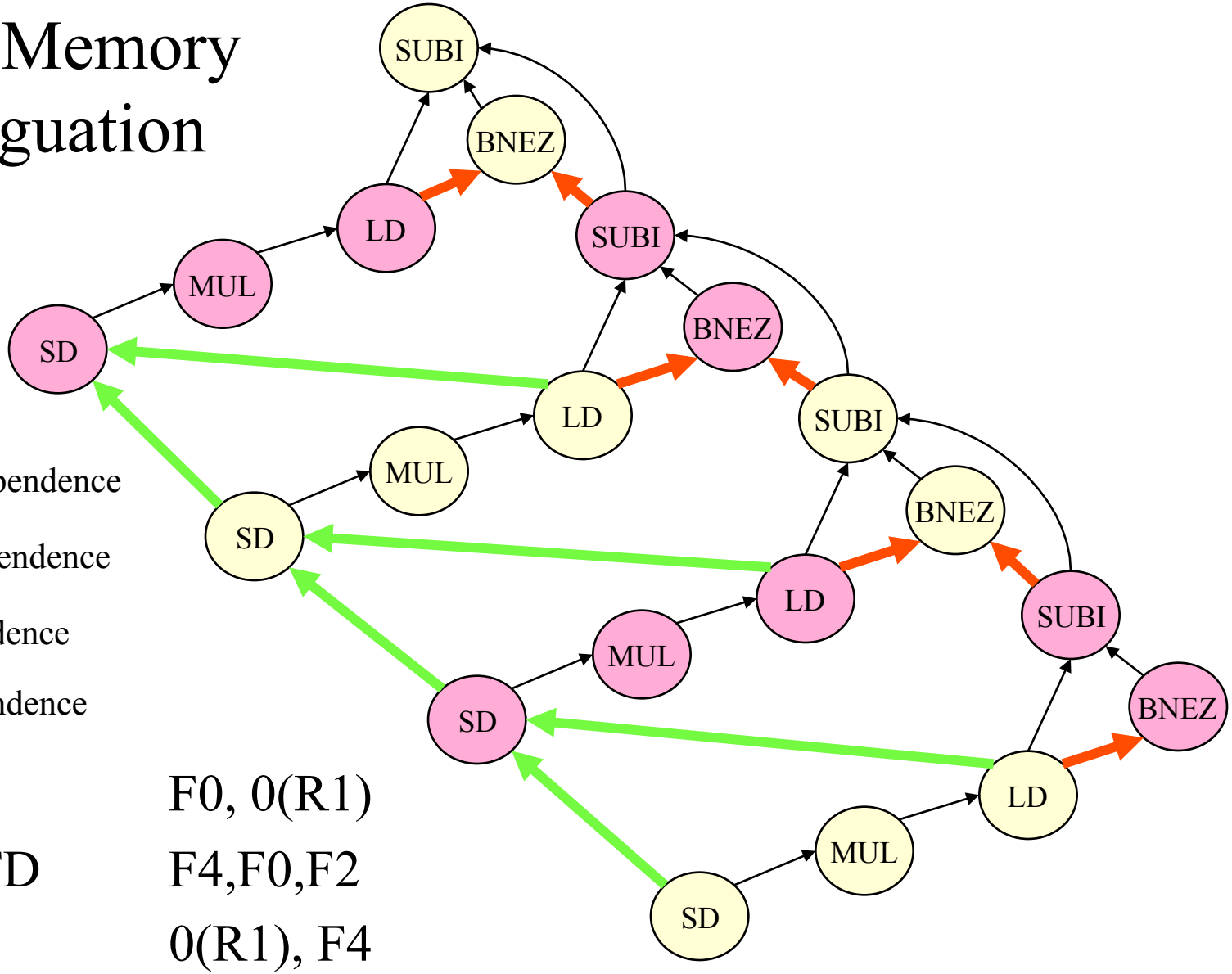
```
Loop:  LD      F0, 0(R1)
       MULTD   F4,F0,F2
       SD      0(R1), F4
       SUBI    R1,R1,#8
       BNEZ    R1, Loop
```

Critical Path = 14




```
Loop:  LD      F0, 0(R1)
       MULTD   F4,F0,F2
       SD      0(R1), F4
       SUBI    R1,R1,#8
       BNEZ    R1, Loop
```

Lets Add Memory Disambiguation



Memory Dependence

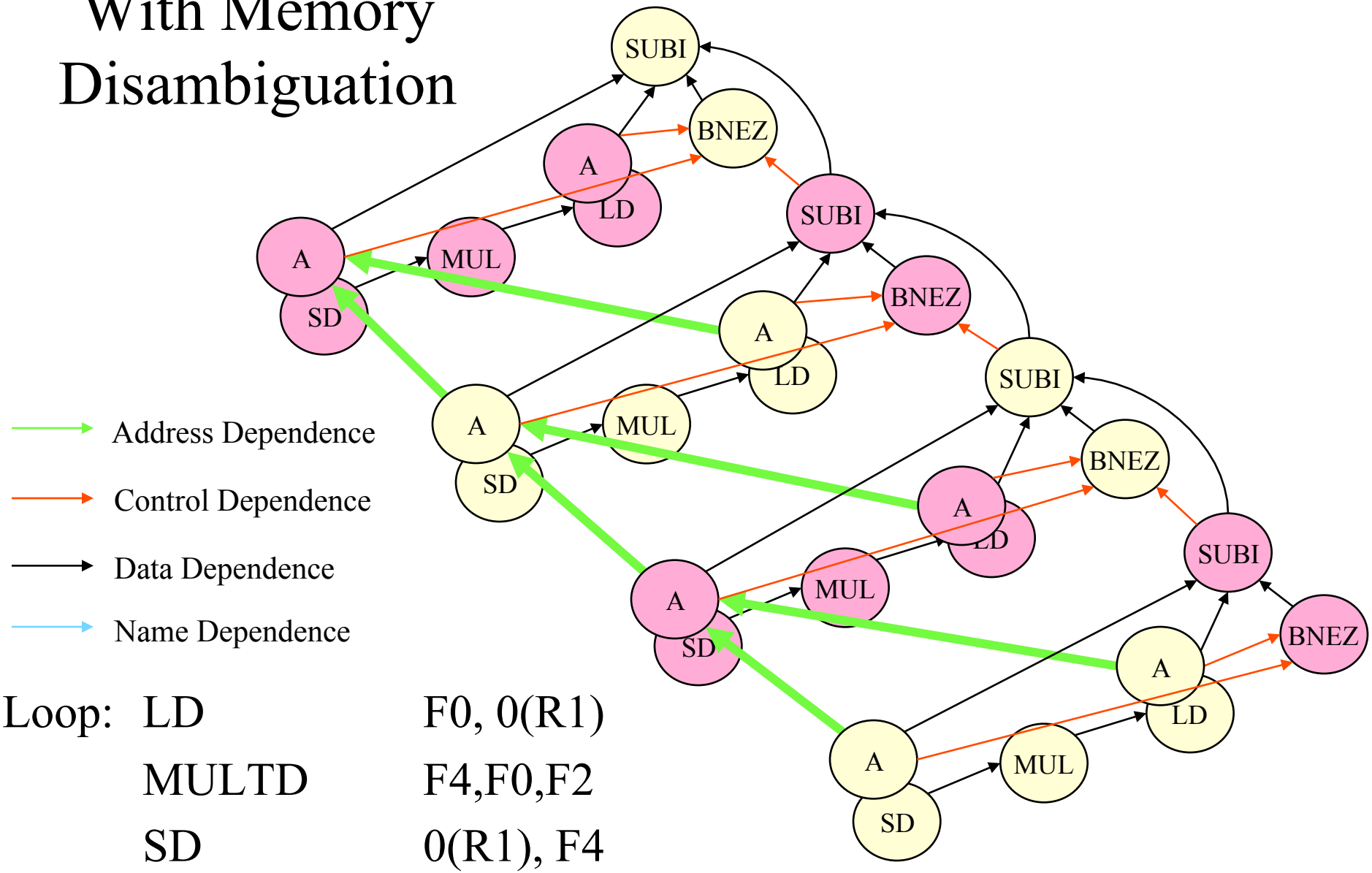
Control Dependence

Data Dependence

Name Dependence

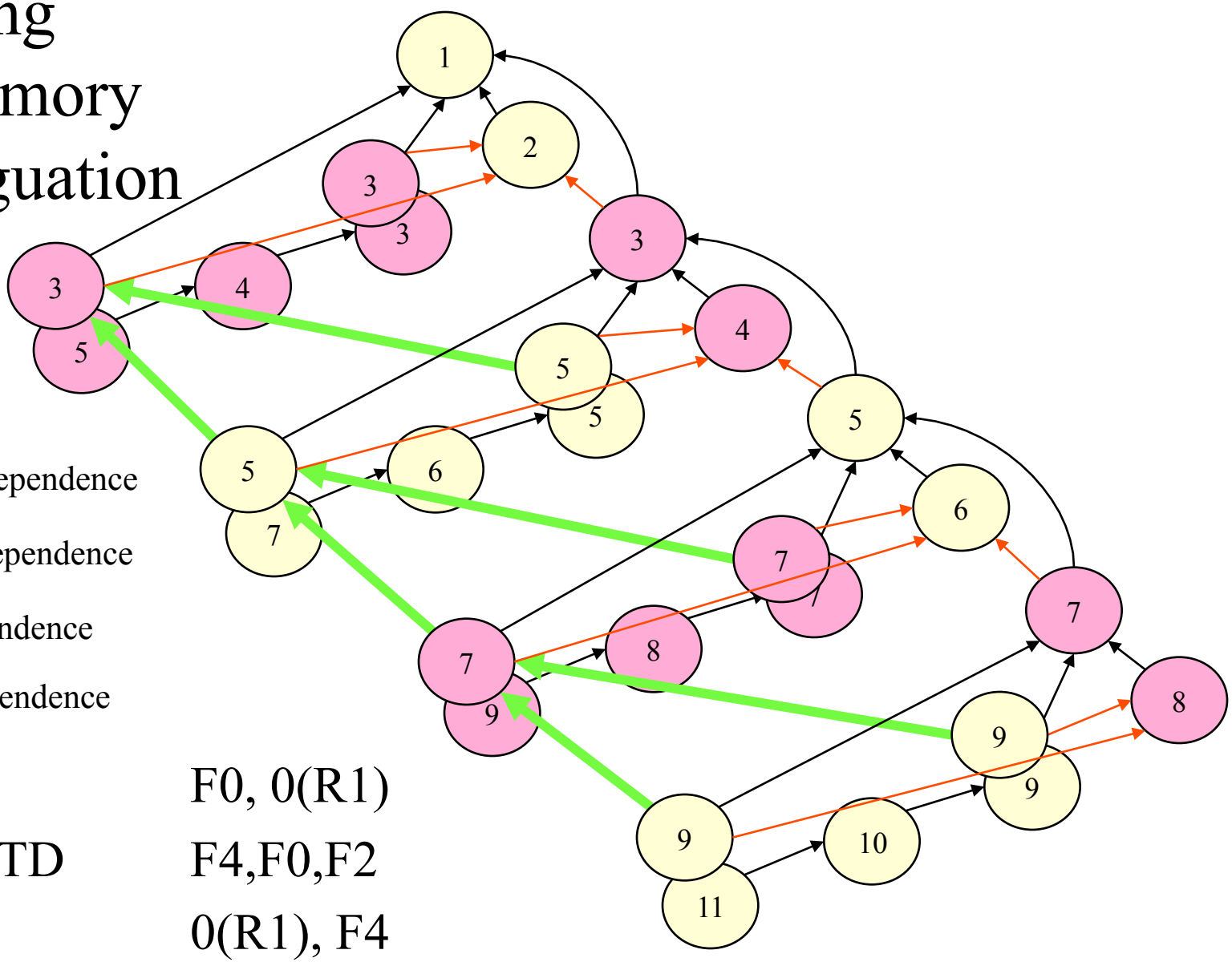
```
Loop:  LD      F0, 0(R1)
       MULTD   F4,F0,F2
       SD      0(R1), F4
       SUBI    R1,R1,#8
       BNEZ    R1, Loop
```

With Memory Disambiguation




```
Loop:  LD      F0, 0(R1)
       MULTD   F4,F0,F2
       SD      0(R1), F4
       SUBI    R1,R1,#8
       BNEZ    R1, Loop
```

Timing with Memory Disambiguation



Address Dependence

Control Dependence

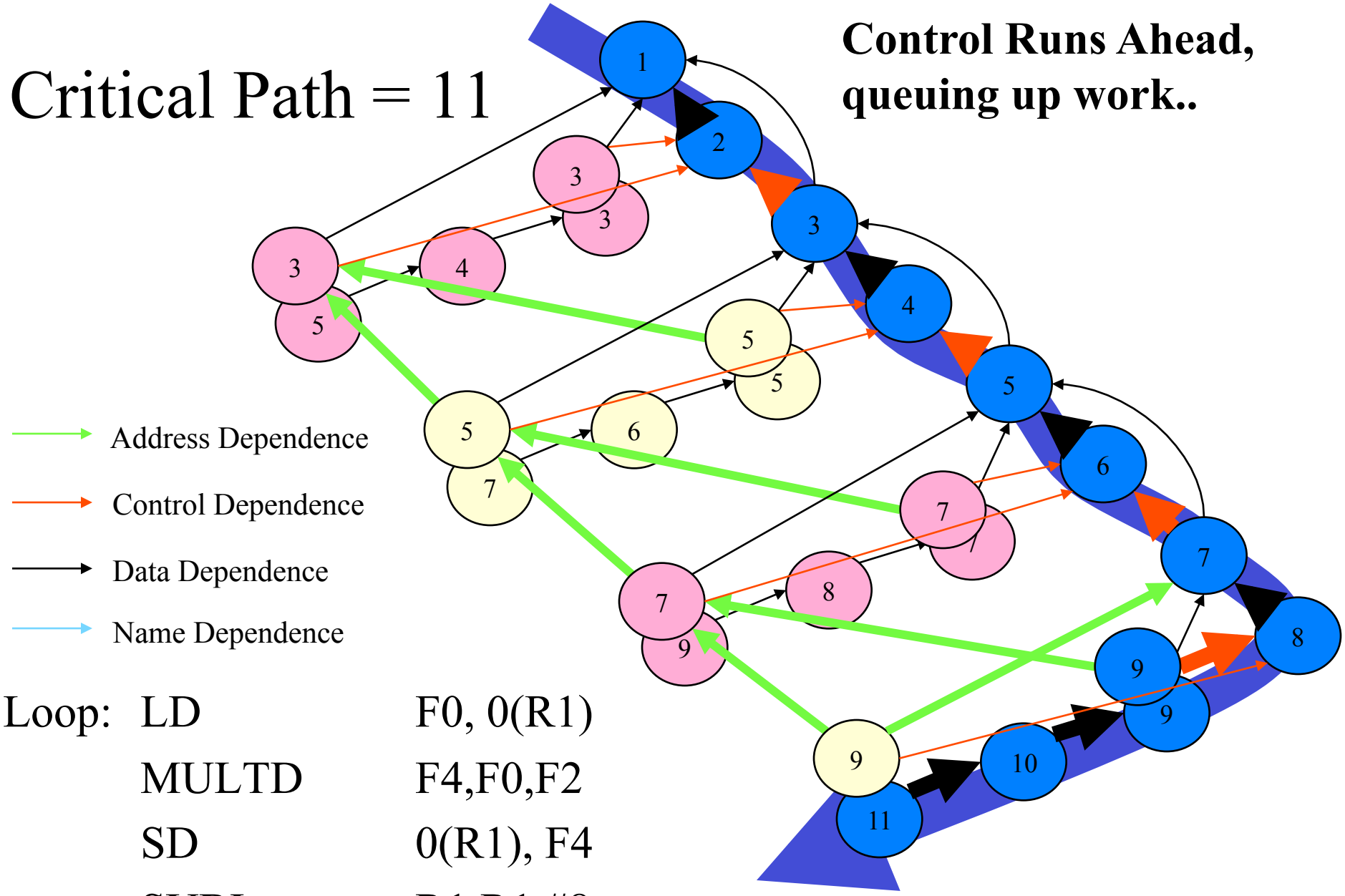
Data Dependence

Name Dependence

```
Loop:  LD      F0, 0(R1)
       MULTD   F4,F0,F2
       SD      0(R1), F4
       SUBI    R1,R1,#8
       BNEZ    R1, Loop
```

Critical Path = 11

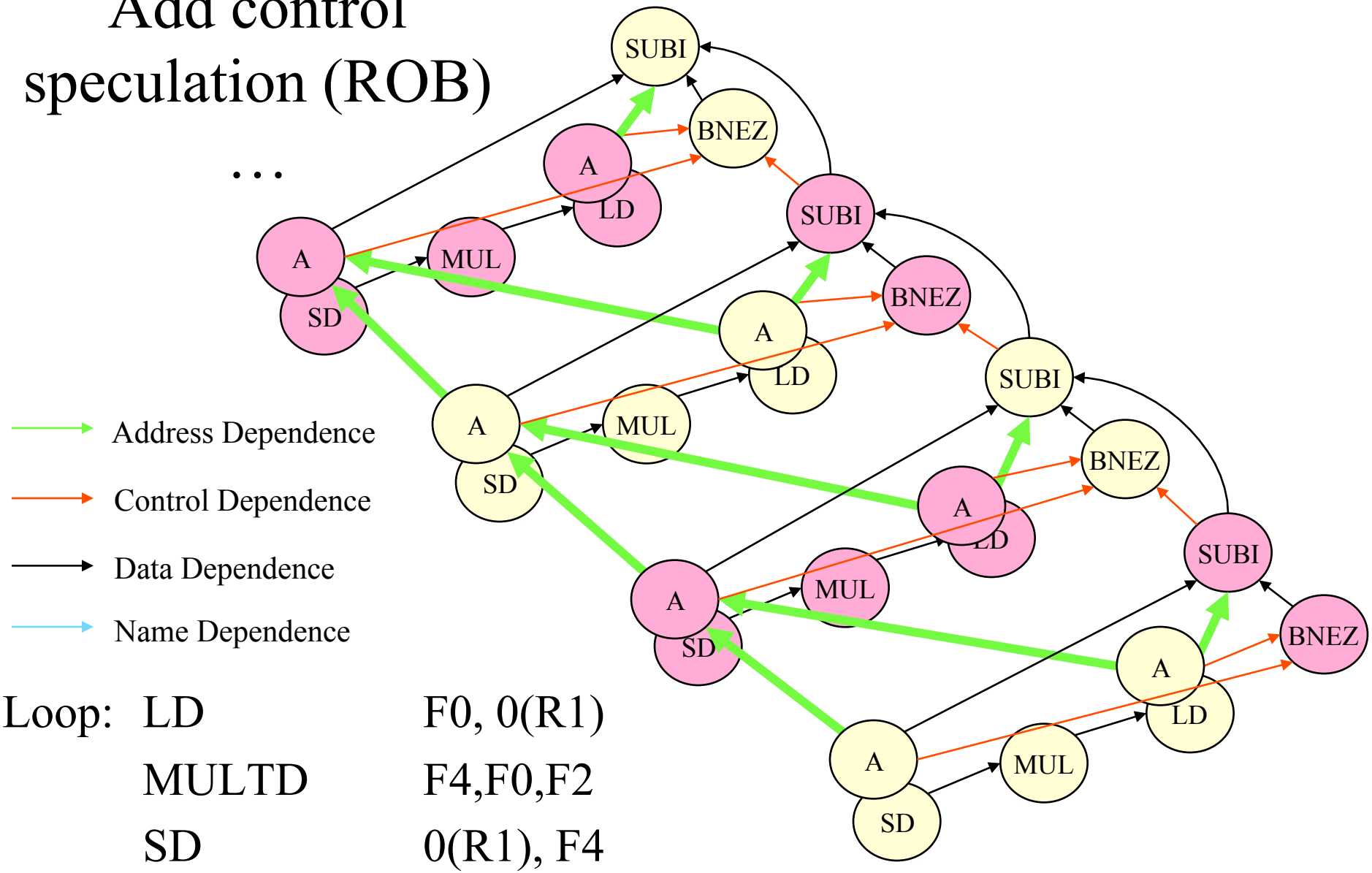
Control Runs Ahead, queuing up work..




```
Loop:  LD      F0, 0(R1)
       MULTD   F4,F0,F2
       SD      0(R1), F4
       SUBI    R1,R1,#8
       BNEZ    R1, Loop
```

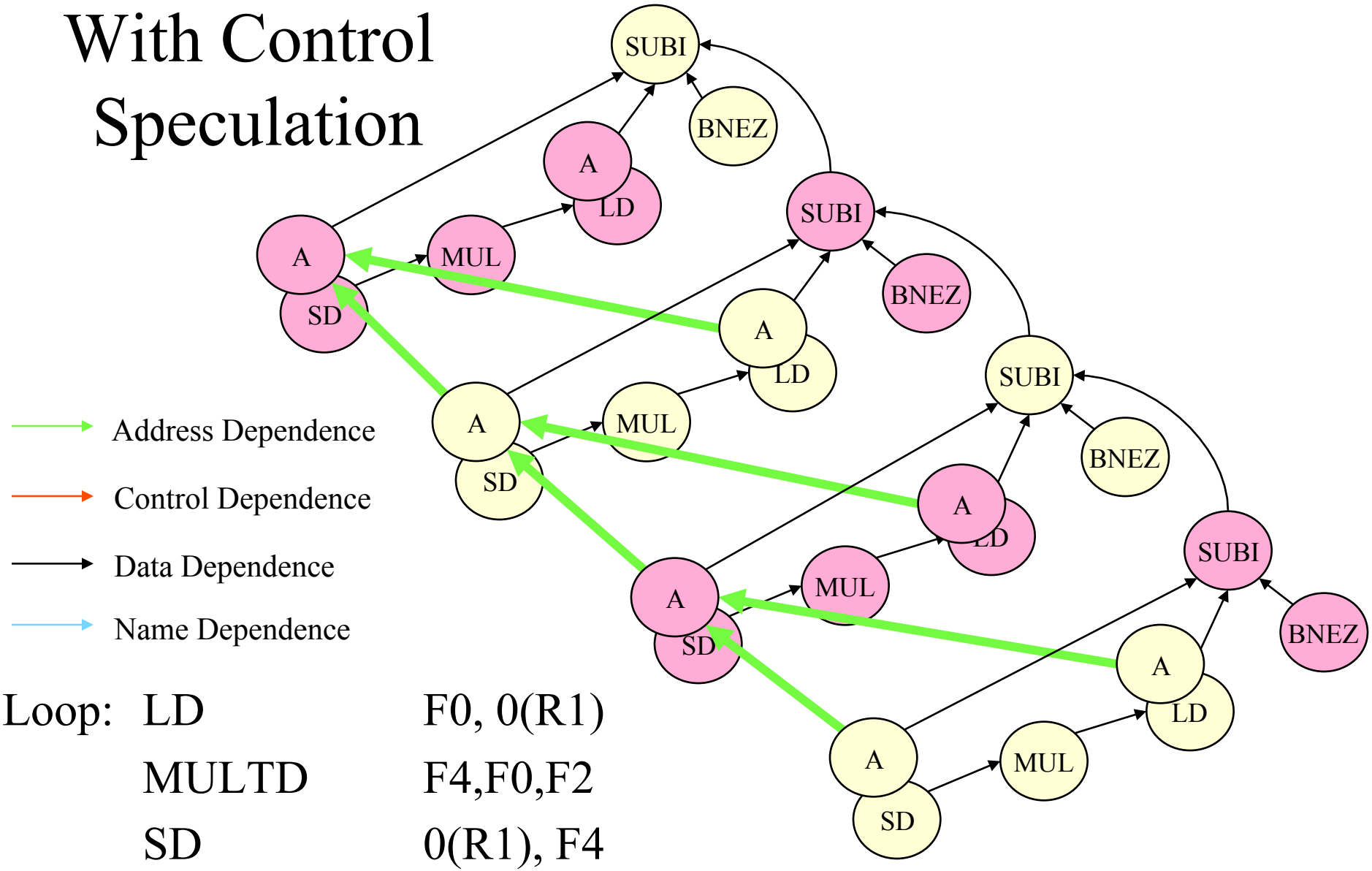
Add control speculation (ROB)

…



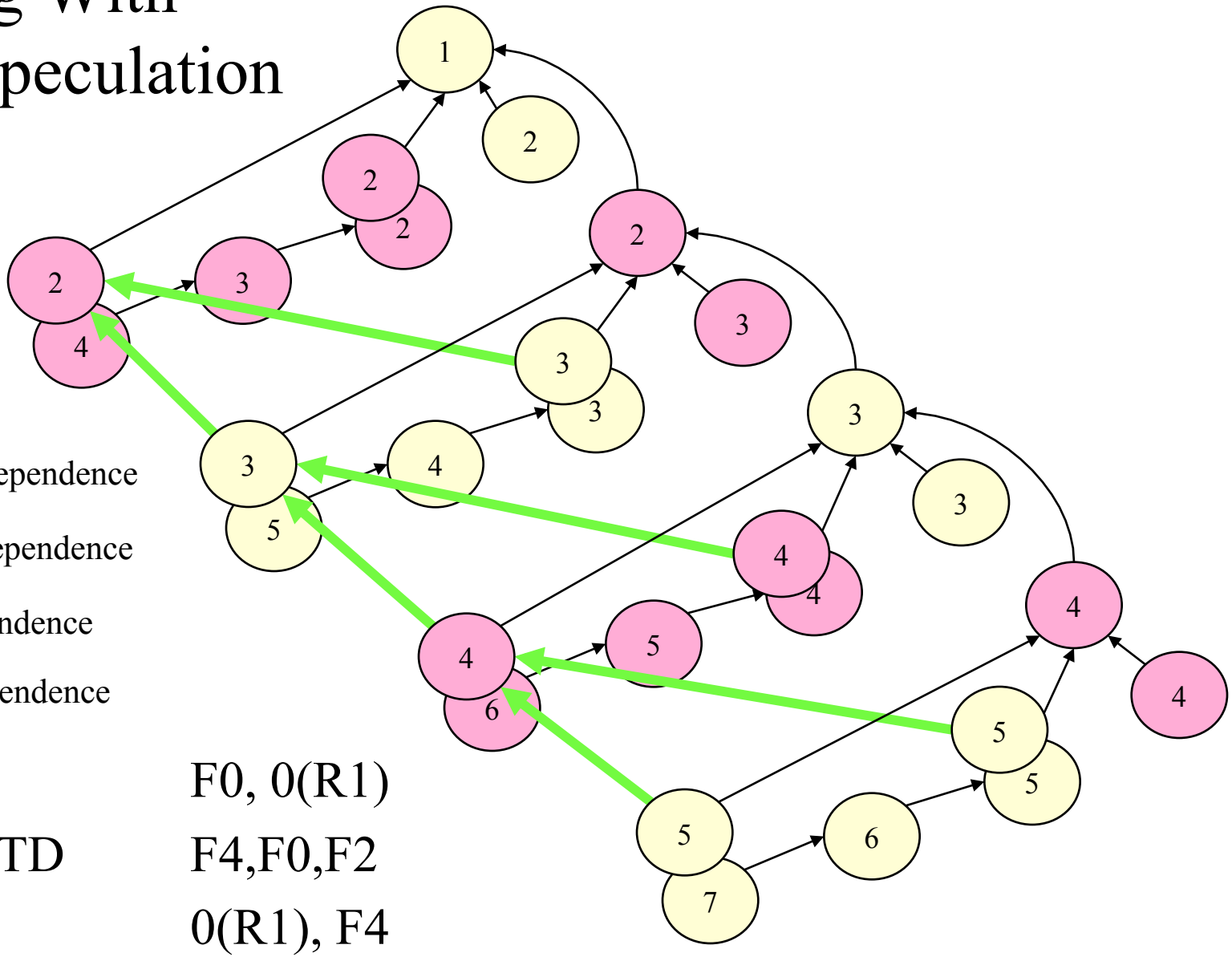

```
Loop:  LD      F0, 0(R1)
       MULTD   F4,F0,F2
       SD      0(R1), F4
       SUBI    R1,R1,#8
       BNEZ    R1, Loop
```

With Control Speculation




```
Loop:  LD      F0, 0(R1)
       MULTD   F4,F0,F2
       SD      0(R1), F4
       SUBI    R1,R1,#8
       BNEZ    R1, Loop
```

Timing With Control Speculation



Address Dependence

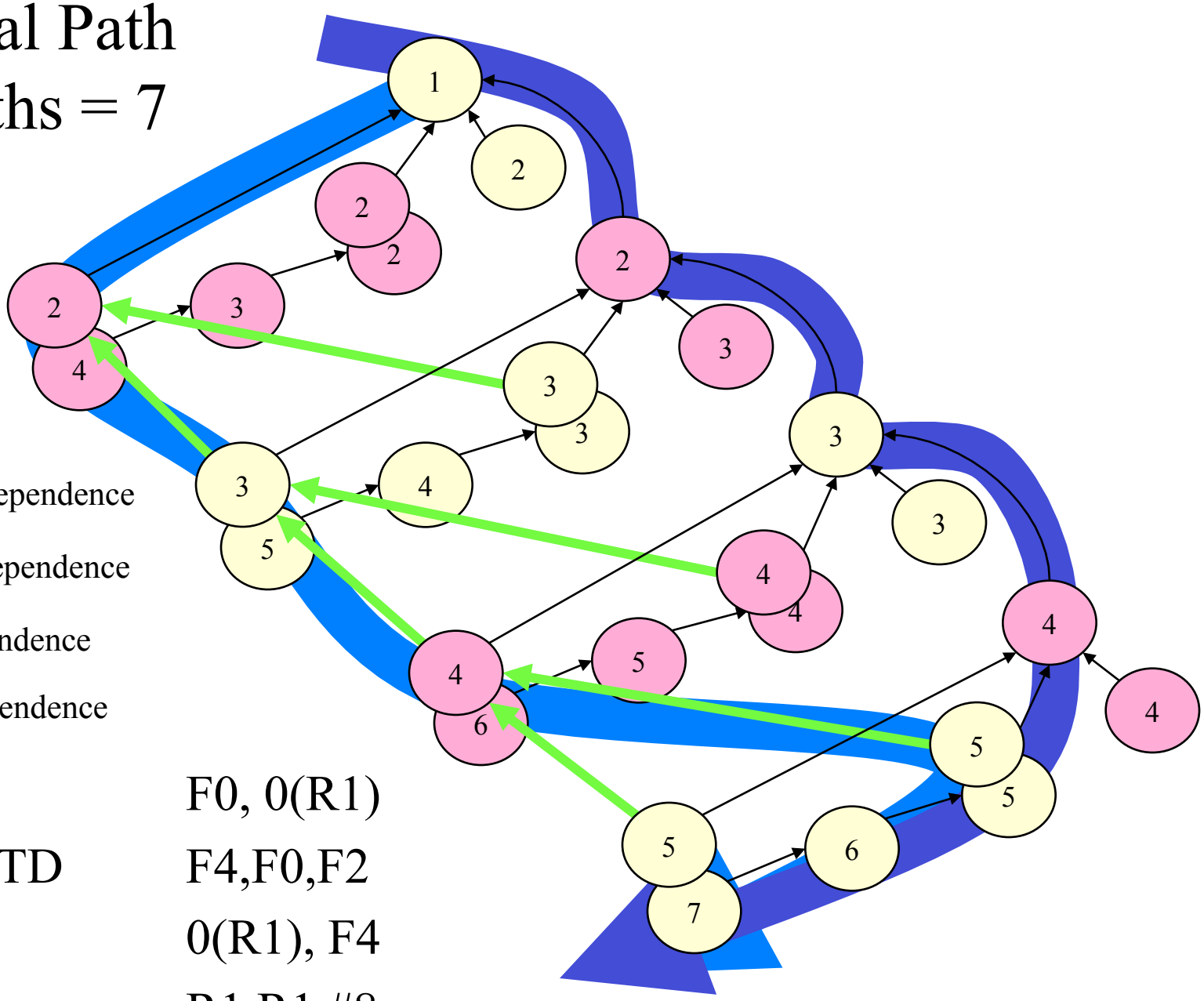
Control Dependence

Data Dependence

Name Dependence

```
Loop:  LD      F0, 0(R1)
       MULTD   F4,F0,F2
       SD      0(R1), F4
       SUBI    R1,R1,#8
       BNEZ    R1, Loop
```

Critical Path
Lengths = 7



Address Dependence

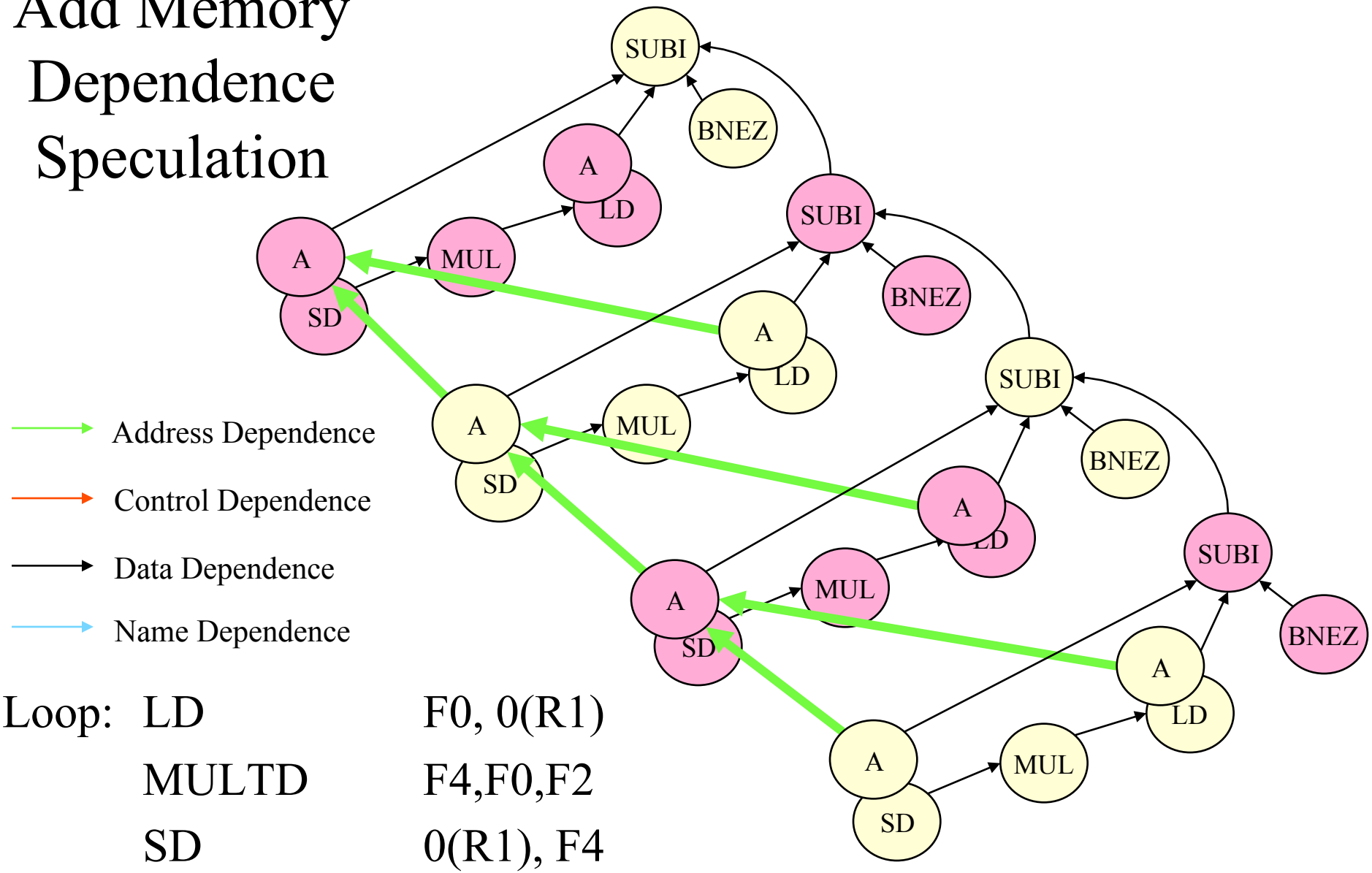
Control Dependence

Data Dependence

Name Dependence

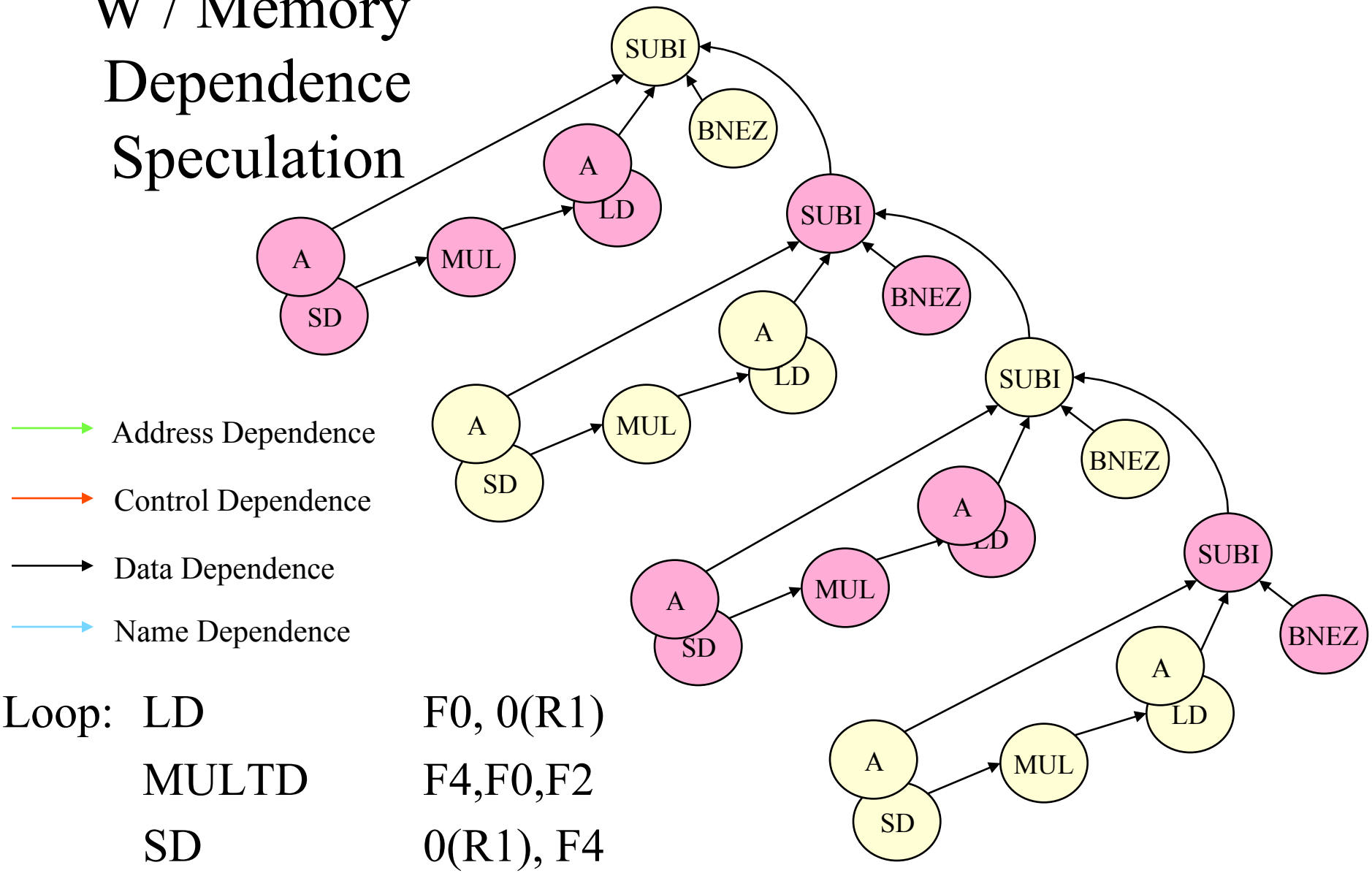
```
Loop:  LD      F0, 0(R1)
       MULTD   F4,F0,F2
       SD      0(R1), F4
       SUBI    R1,R1,#8
       BNEZ    R1, Loop
```

Add Memory Dependence Speculation



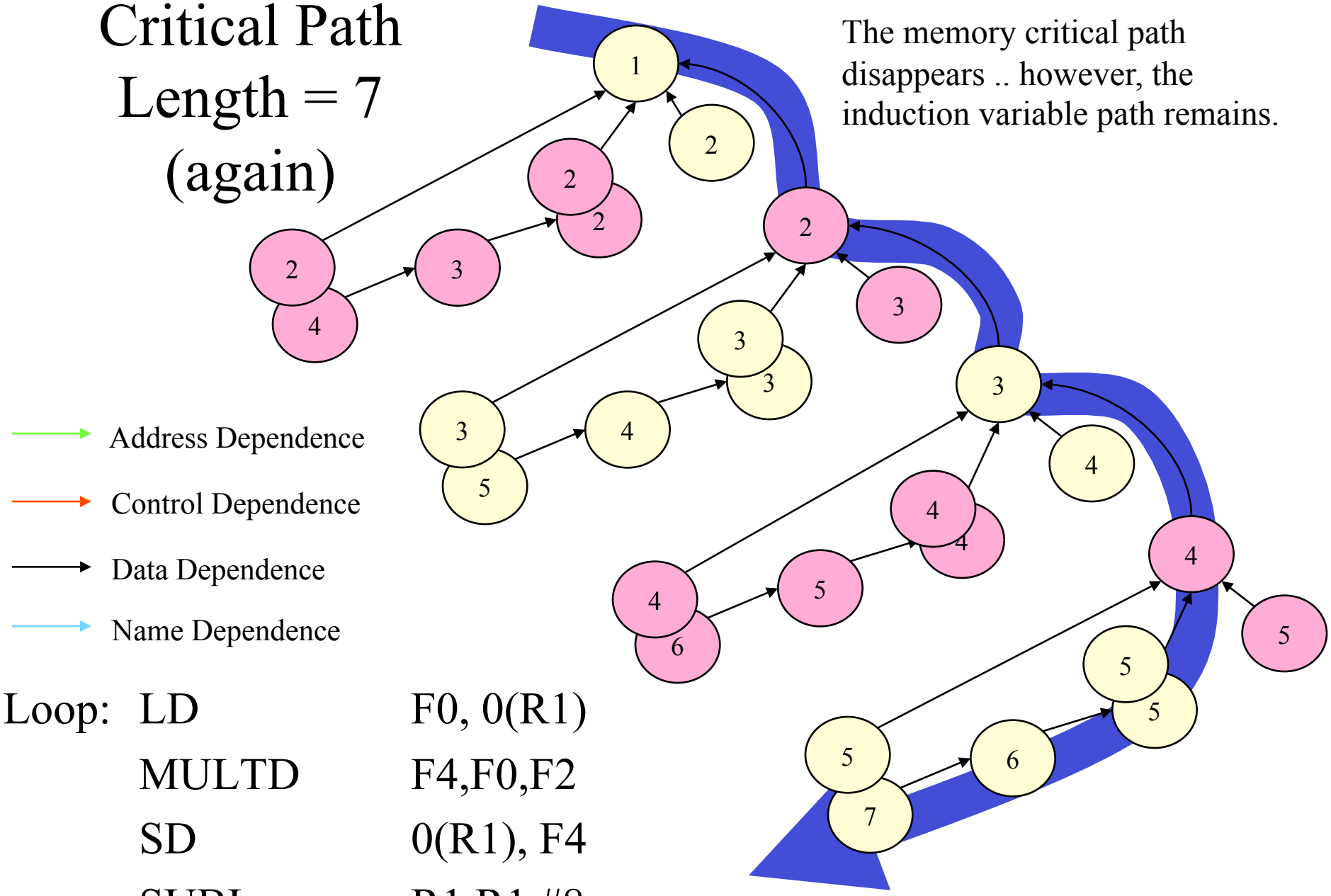

```
Loop:  LD      F0, 0(R1)
       MULTD   F4,F0,F2
       SD      0(R1), F4
       SUBI    R1,R1,#8
       BNEZ    R1, Loop
```

W / Memory Dependence Speculation




```
Loop:  LD      F0, 0(R1)
       MULTD   F4,F0,F2
       SD      0(R1), F4
       SUBI    R1,R1,#8
       BNEZ    R1, Loop
```

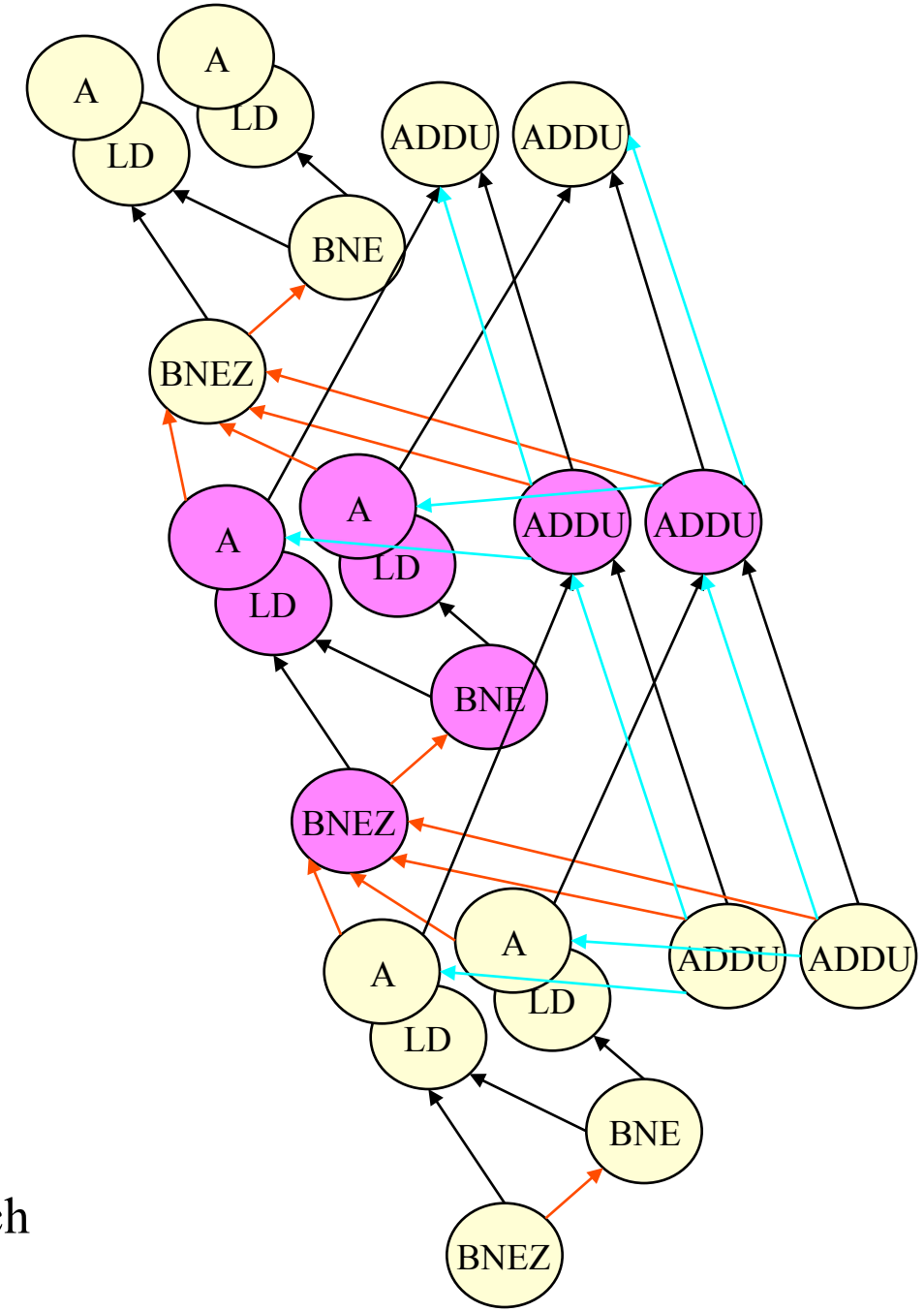
Critical Path Length = 7 (again)



The memory critical path disappears .. however, the induction variable path remains.

```
Loop:  LD      F0, 0(R1)
       MULTD   F4,F0,F2
       SD      0(R1), F4
       SUBI    R1,R1,#8
       BNEZ    R1, Loop
```

A New Loop



Address Dependence

Control Dependence

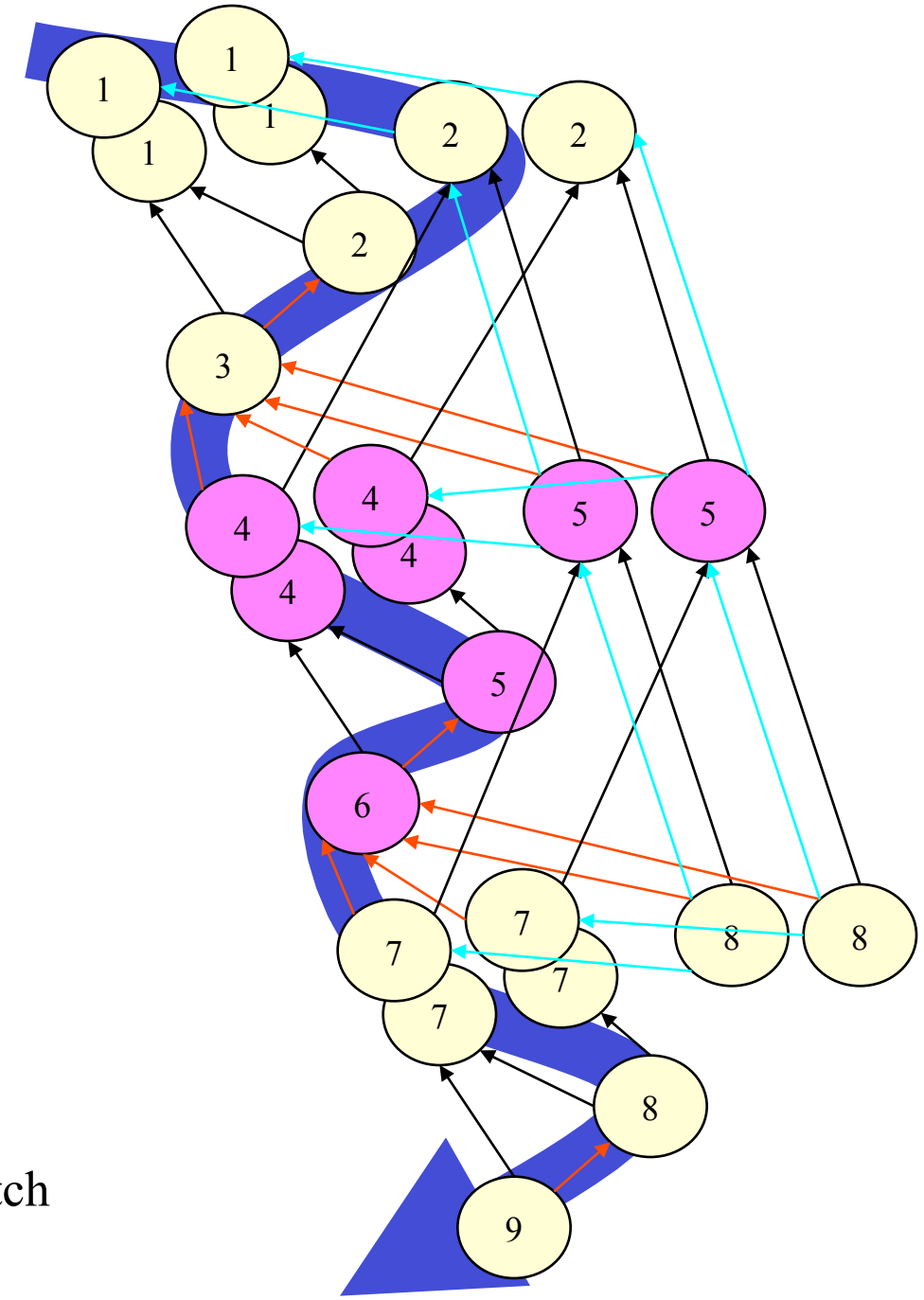
Data Dependence

Name Dependence

```
Loop:   LB      R4, 0(R1)
        LB      R5, 0(R2)
        ADDU    R1,R1,1
        ADDU    R2,R2,1
        BNE     R4,R5, Mismatch
        BNEZ    R4, Loop
```

what is this code? Is it parallel?

A New Loop



Address Dependence

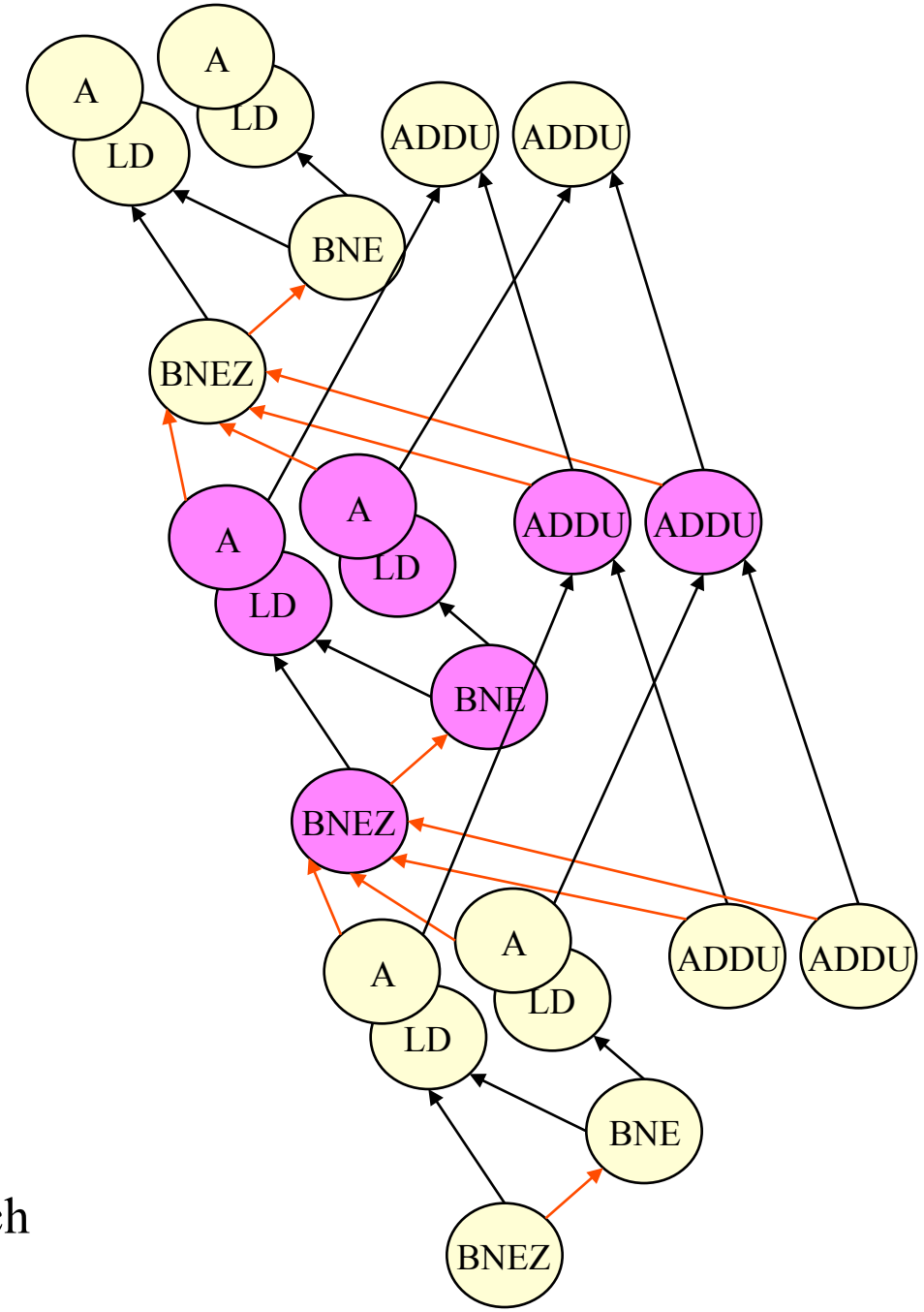
Control Dependence

Data Dependence

Name Dependence

```
Loop:   LW      R4, 0(R1)
        LW      R5, 0(R2)
        ADDU    R1,R1,4
        ADDU    R2,R2,4
        BNE     R4,R5, Mismatch
        BNEZ    R4, Loop
```

Renaming
+
Dynamic Scheduling



Address Dependence

Control Dependence

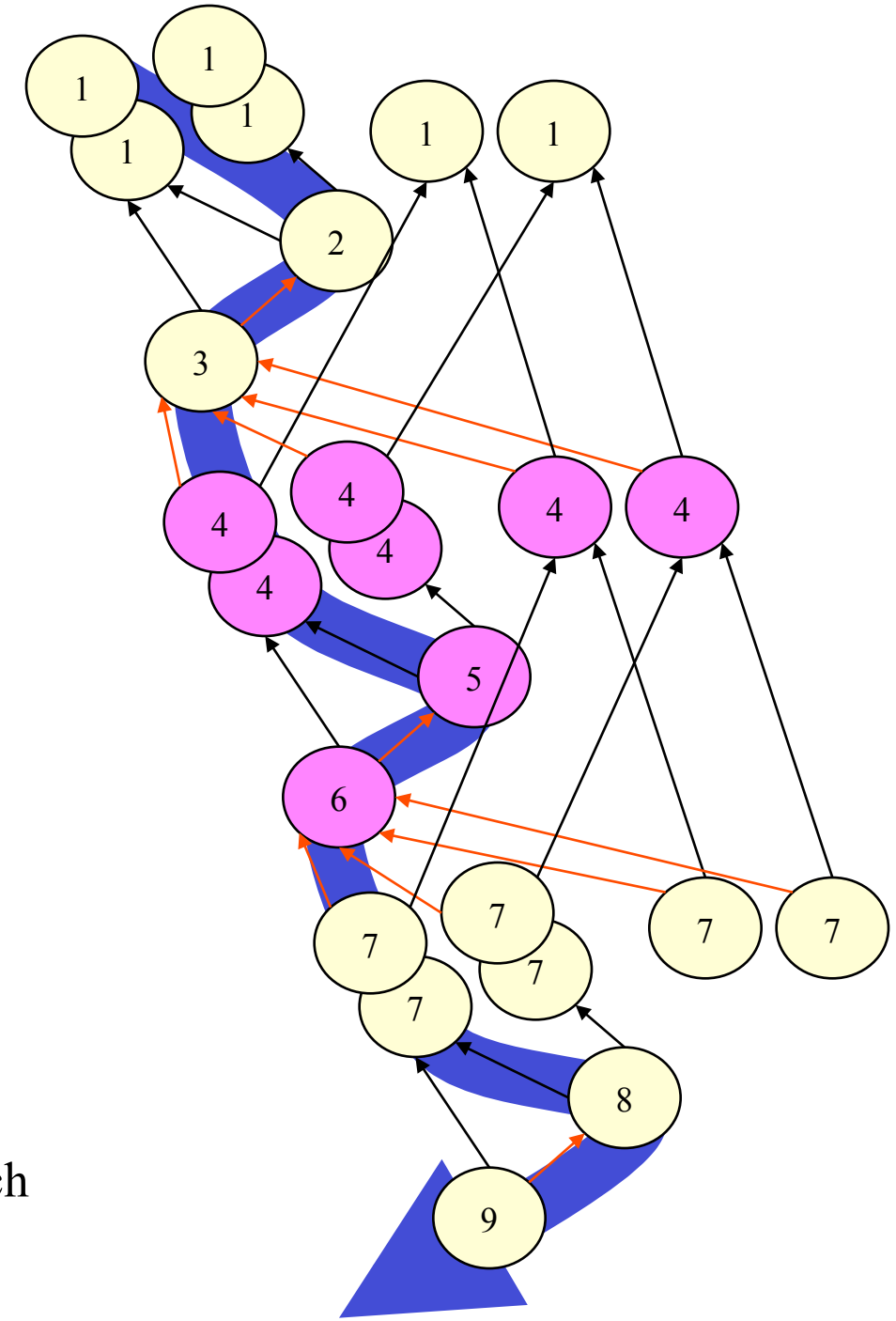
Data Dependence

Name Dependence

```
Loop:   LW      R4, 0(R1)
        LW      R5, 0(R2)
        ADDU    R1,R1,4
        ADDU    R2,R2,4
        BNE     R4,R5, Mismatch
        BNEZ    R4, Loop
```

Renaming, Dynamic Scheduling:

No overlap of loop iterations



Address Dependence

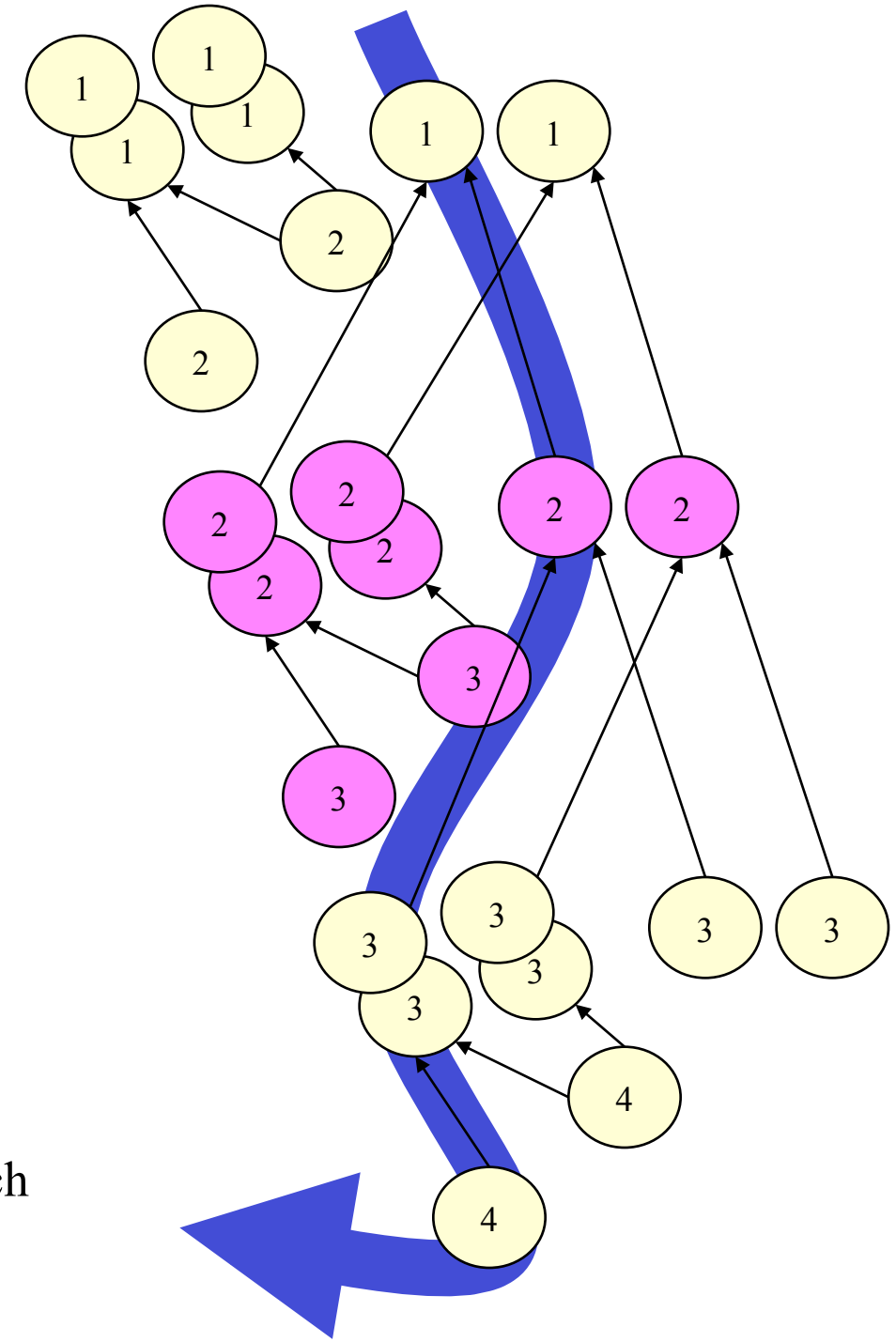
Control Dependence

Data Dependence

Name Dependence

```
Loop:   LW      R4, 0(R1)
        LW      R5, 0(R2)
        ADDU    R1,R1,4
        ADDU    R2,R2,4
        BNE     R4,R5, Mismatch
        BNEZ    R4, Loop
```

Renaming, Dynamic Scheduling, Speculation - Allows Overlap



Address Dependence

Control Dependence

Data Dependence

Name Dependence

```
Loop:   LW      R4, 0(R1)
        LW      R5, 0(R2)
        ADDU    R1,R1,4
        ADDU    R2,R2,4
        BNE     R4,R5, Mismatch
        BNEZ    R4, Loop
```