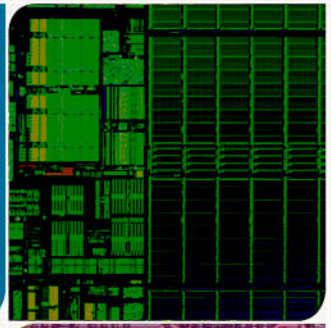
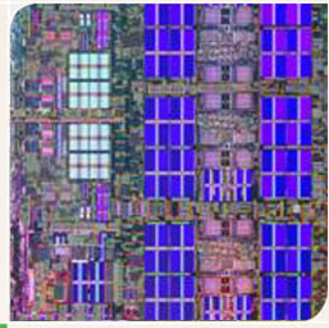
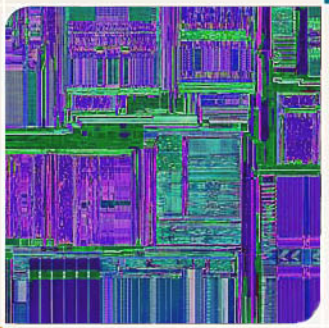
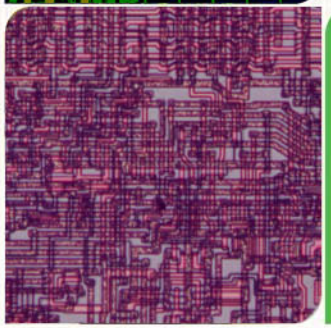
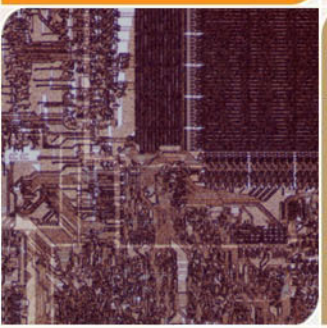







CS 4290/6290

Fall 2011

Prof. Hyesoon Kim

Georgia Tech College of Computing

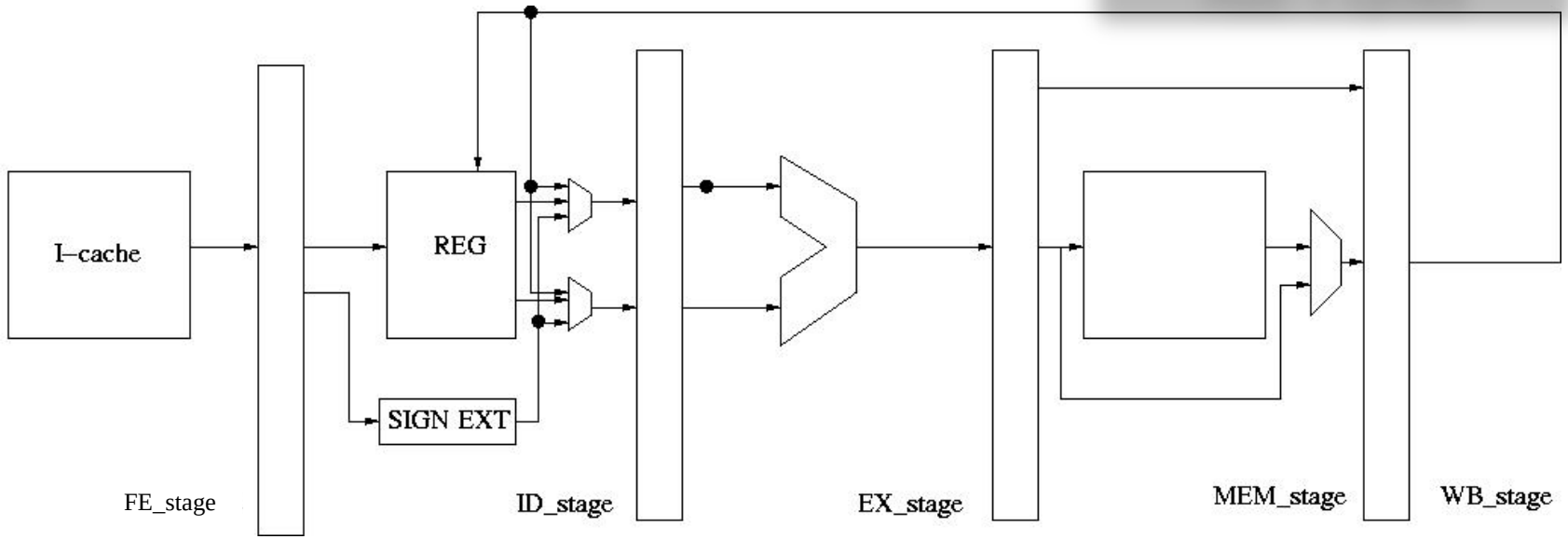


Review

- App1 IPC: 1.5 App2 IPC 0.5
  - Average IPC ?

Dependent Instructions: dst data is available at WB

Add: 1 cycles

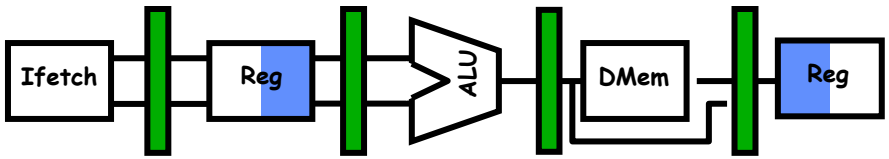



```
add  r1, r2, r3
sub  r4, r1, r3
mul  r5, r2, r3
```

```
FE  L   ID  L   EX  L   MEM  L   WB  L
add     add
sub     sub     add     add
mul     sub     sub             add     add
mul     sub     sub                     add     add
mul     mul     sub     sub                     add
        mul     mul     sub     sub
```

Dependences/Dependencies

- Data Dependencies
  - RAW: Read-After-Write (True Dependence)
  - WAR: Anti-Depedence
  - WAW: Output Dependence
- Control Dependence
  - When following instructions depend on the outcome of a previous branch/jump





Data Dependencies

- Register dependencies
  - RAW, WAR, WAW, based on register *number*
- Memory dependencies



Memory Dependencies

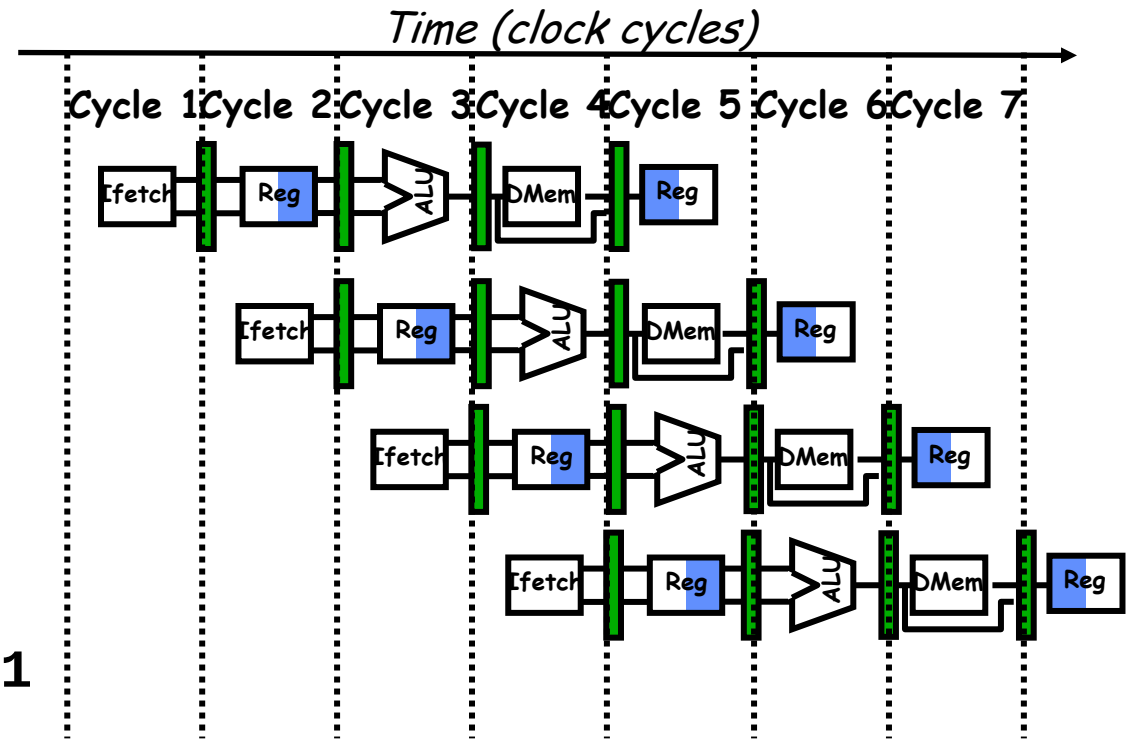
- Basically similar to regular (register) data dependencies: RAW, WAR, WAW
- However, the exact location is not known until run-time :
  - A: STORE R1, 0[R2]
  - B: LOAD R5, 24[R8]
  - C: STORE R3, -8[R9]

  - RAW exists if (R2+0) == (R8+24)
  - WAR exists if (R8+24) == (R9 – 8)
  - WAW exists if (R2+0) == (R9 – 8)

Control Hazard

Control hazard

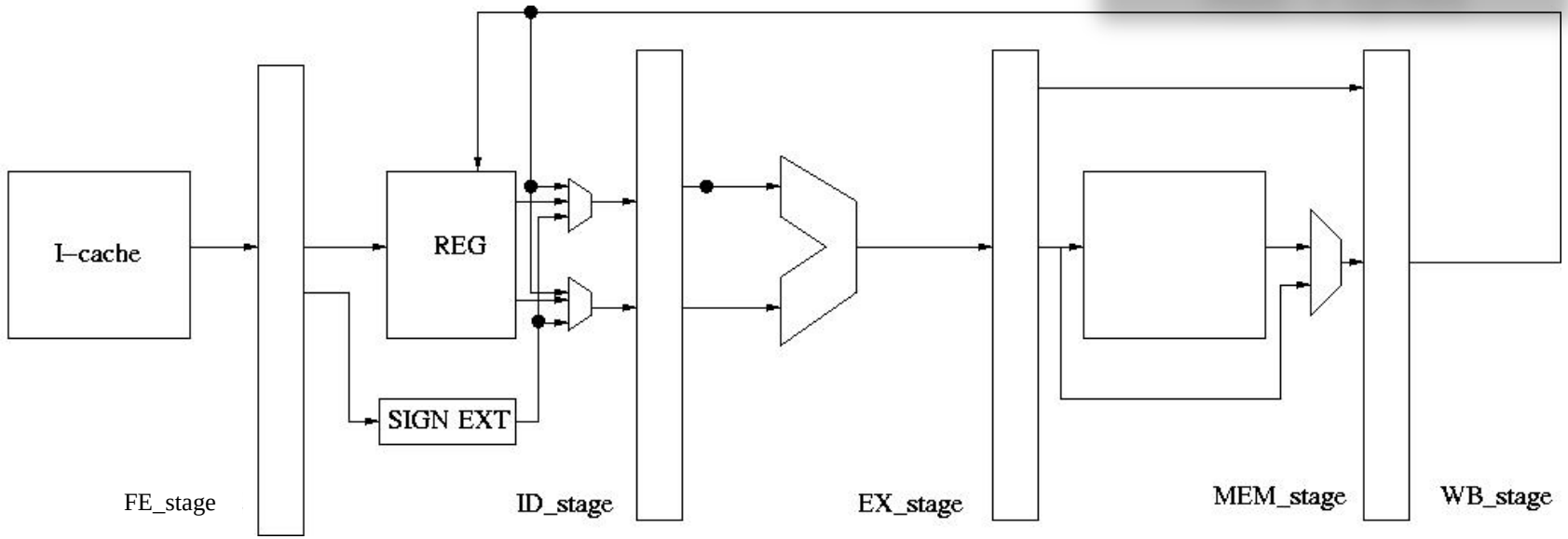
```
Instr. Order
    add r1,r2,r3
    uncond. TARG
    and r6,r1,r7
TARG
    or   r1,r2,r9
    br (r1) TARG1
```



Control hazard and data hazard

Dependent Instructions: dst data is available at WB

Add: 1 cycles




```
add  r1, r2, r3
sub  r4, r1, r3
mul  r5, r2, r3
```

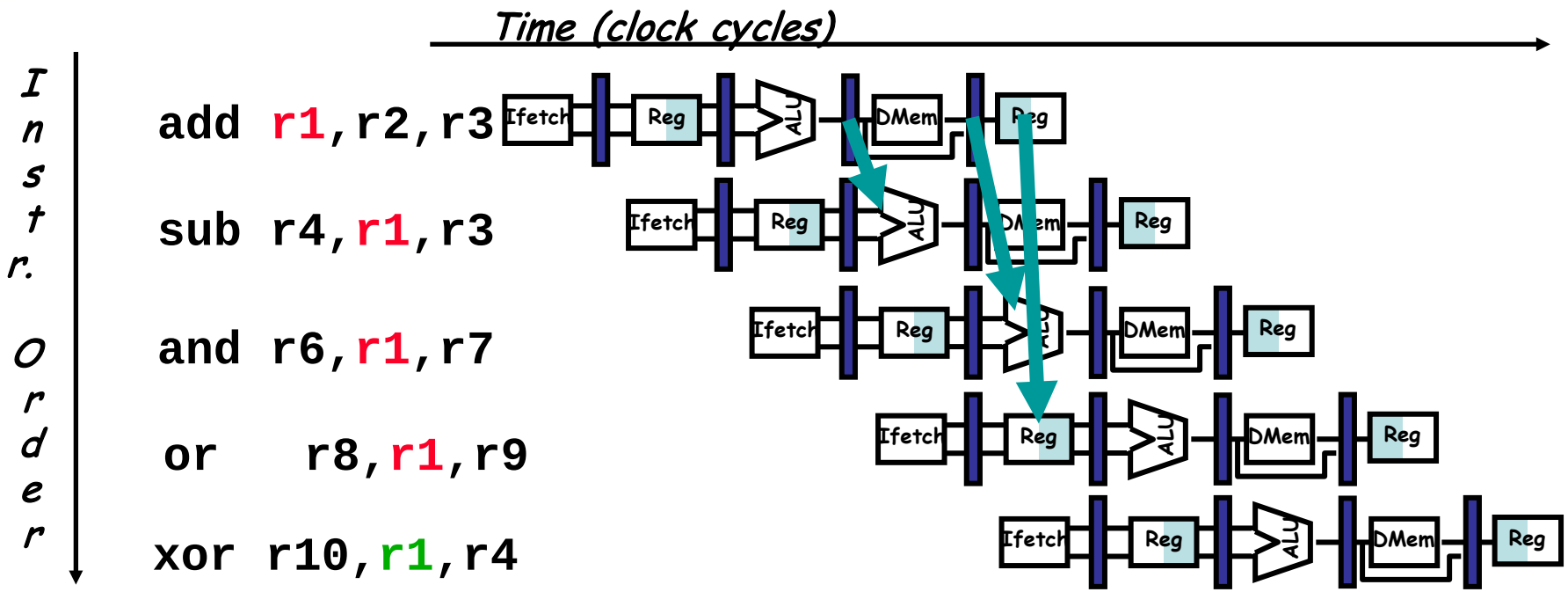
FE L ID L EX L MEM L WB L

```
add  add
sub  sub  add  add
mul  sub  sub  add  add
mul  sub  sub  add  add
mul  mul  sub  sub  add
     mul  mul  sub  sub
```

Can we do better than this?

Georgia Tech College of Computing

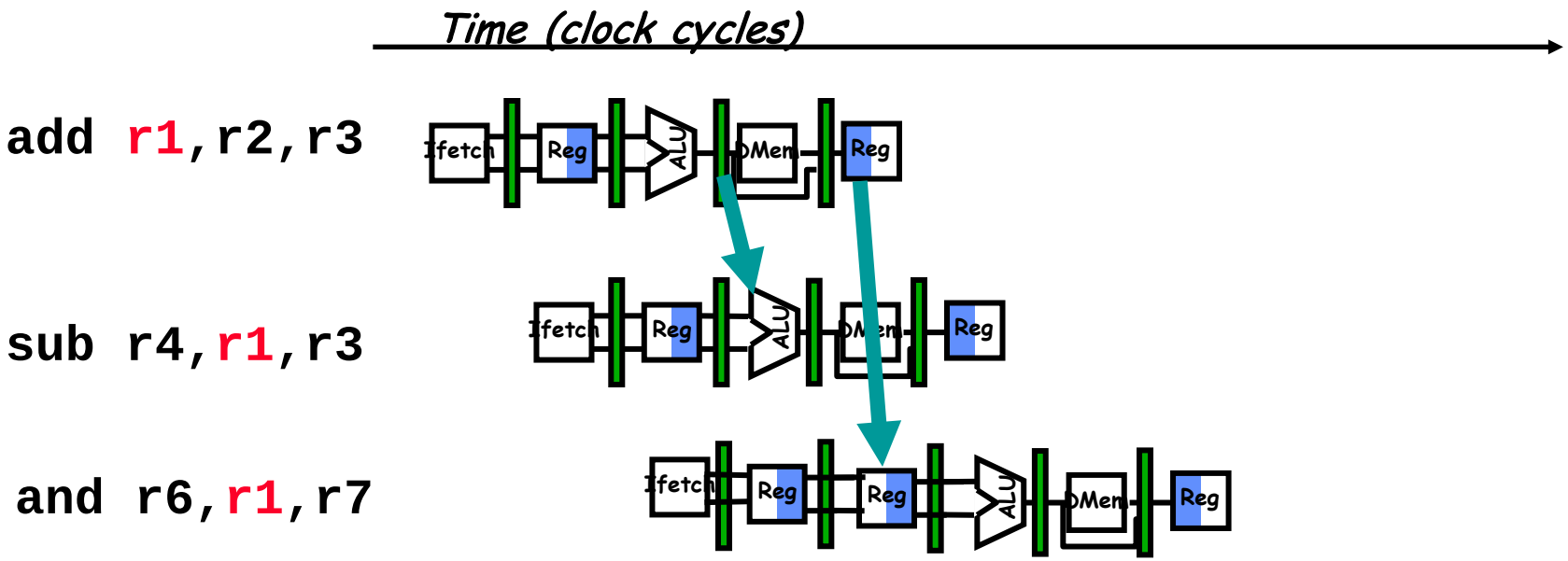
Data Forwarding



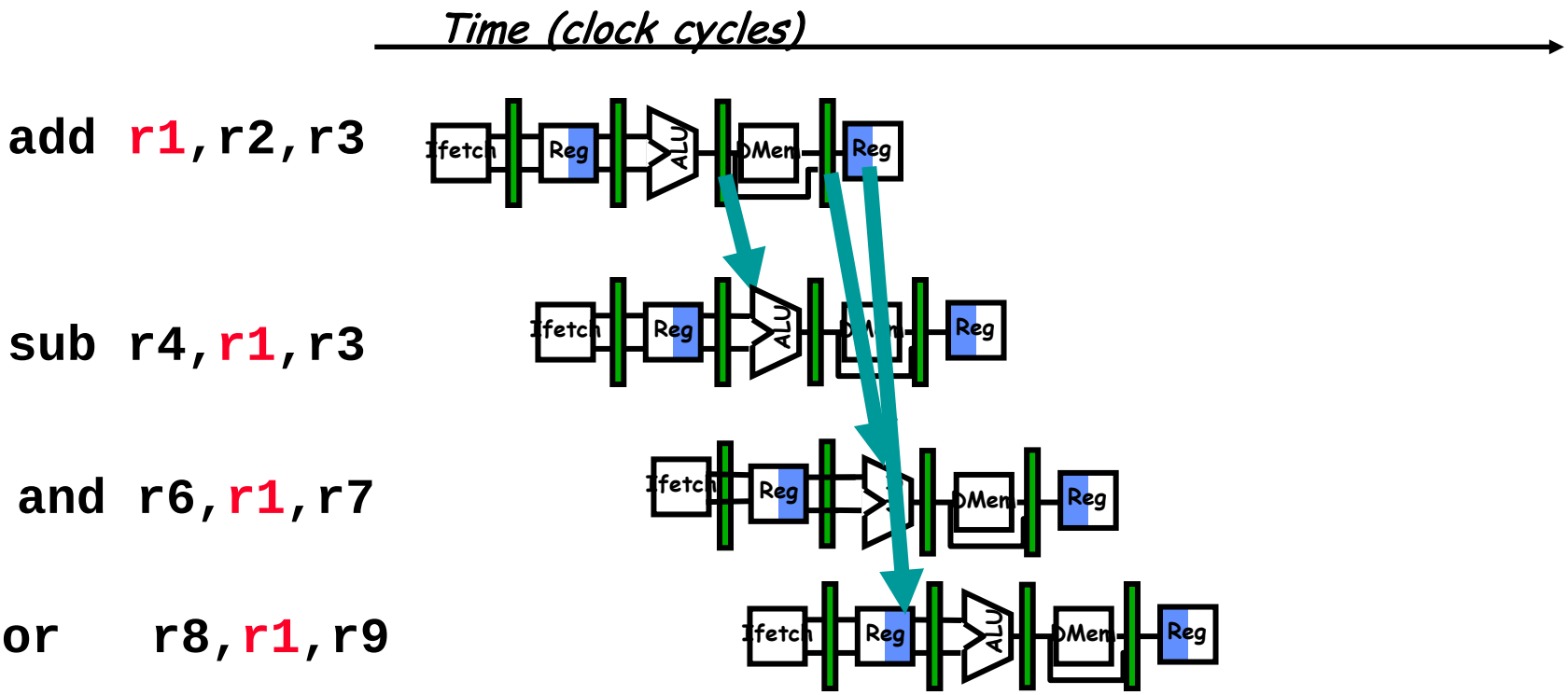
- Destination register is a name for instr's result
- Source registers are names for sources
- Forwarding logic builds data dependence (flow) graph for instructions in the execution window

W/O Forwarding

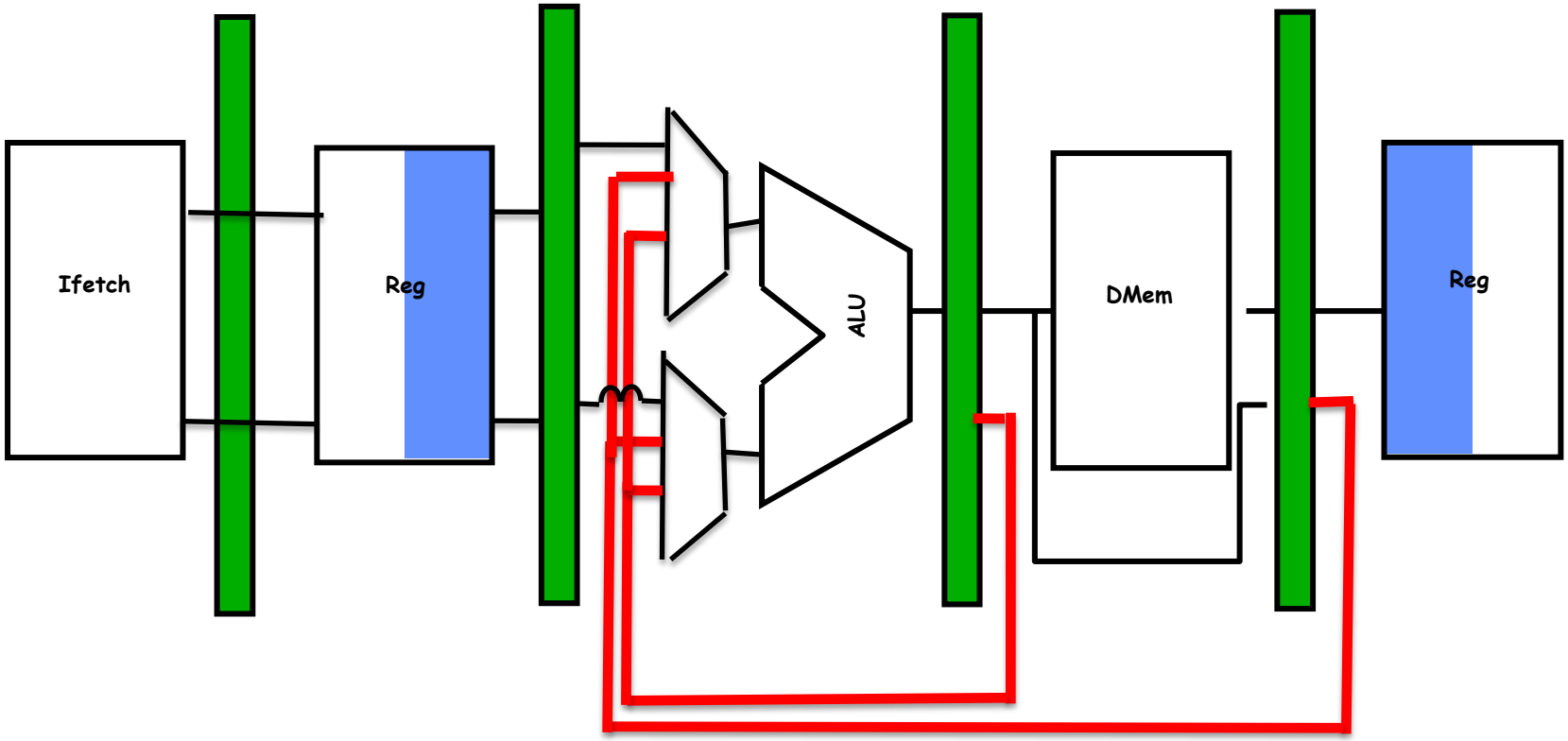
With One Forwarding path



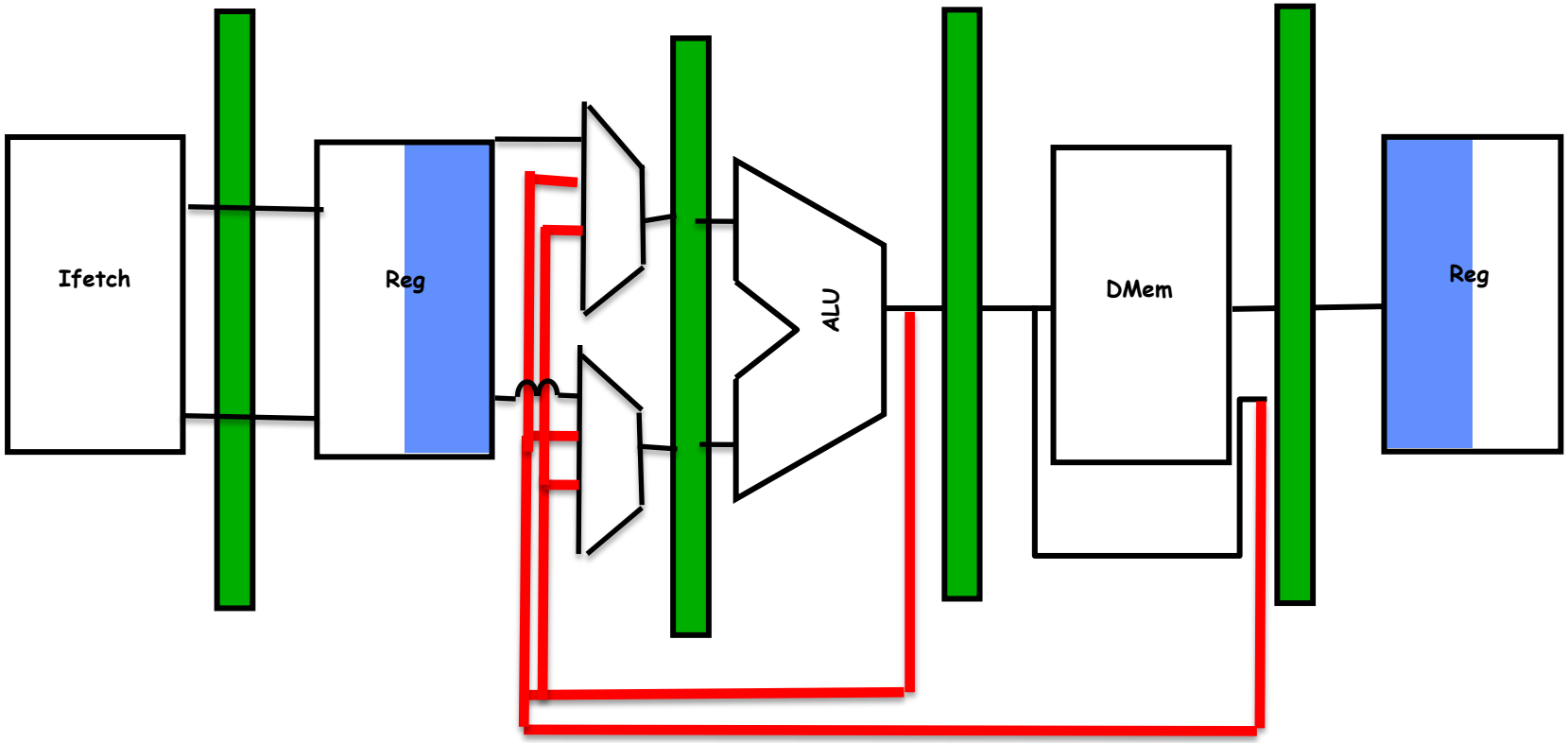
With Two Forwarding paths



Data forwarding Logic (example 1)



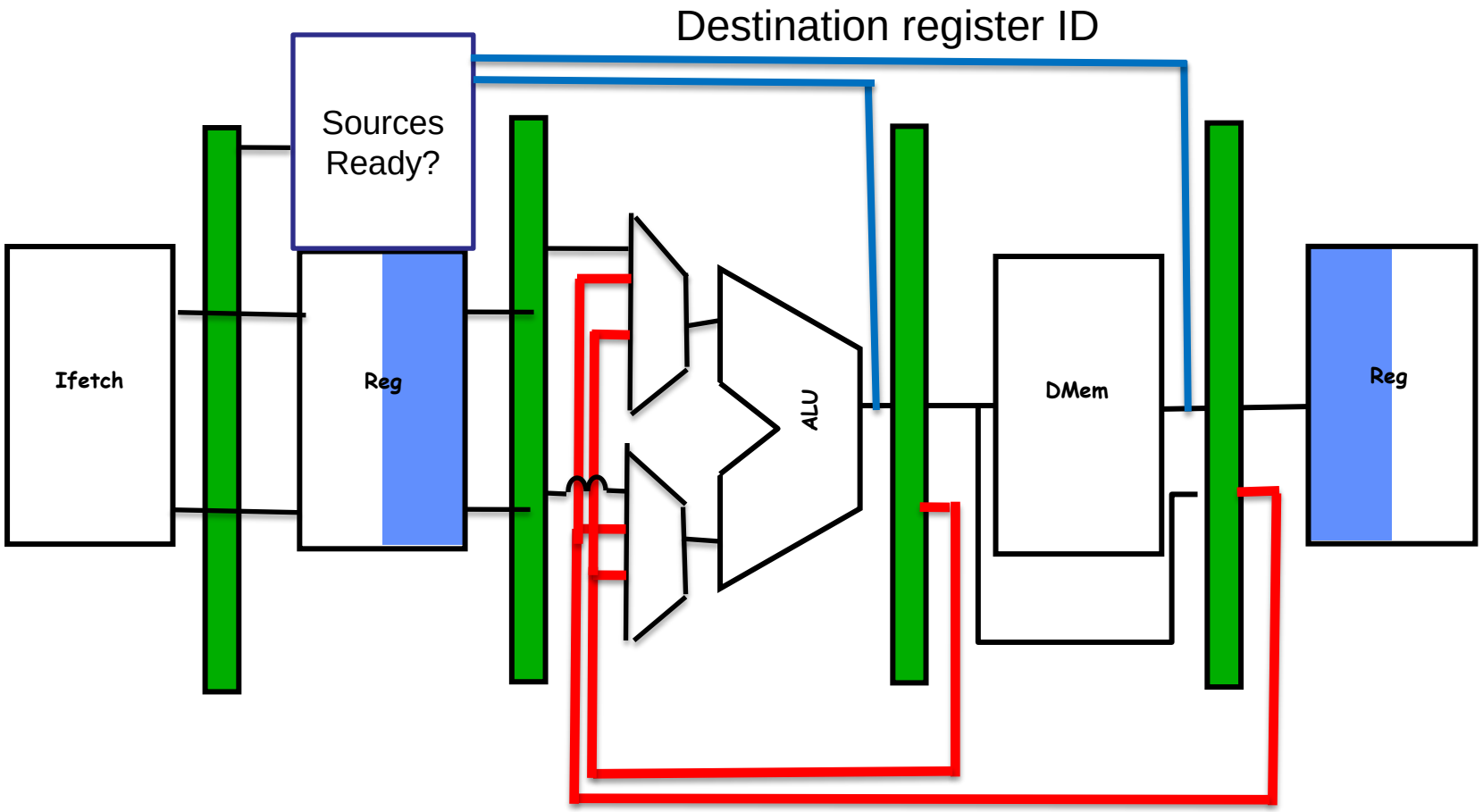
Data forwarding Logic (example 2)





- Is that all?
- How Do we know whether Add is writing a data to R1?
- Broadcasting Register id

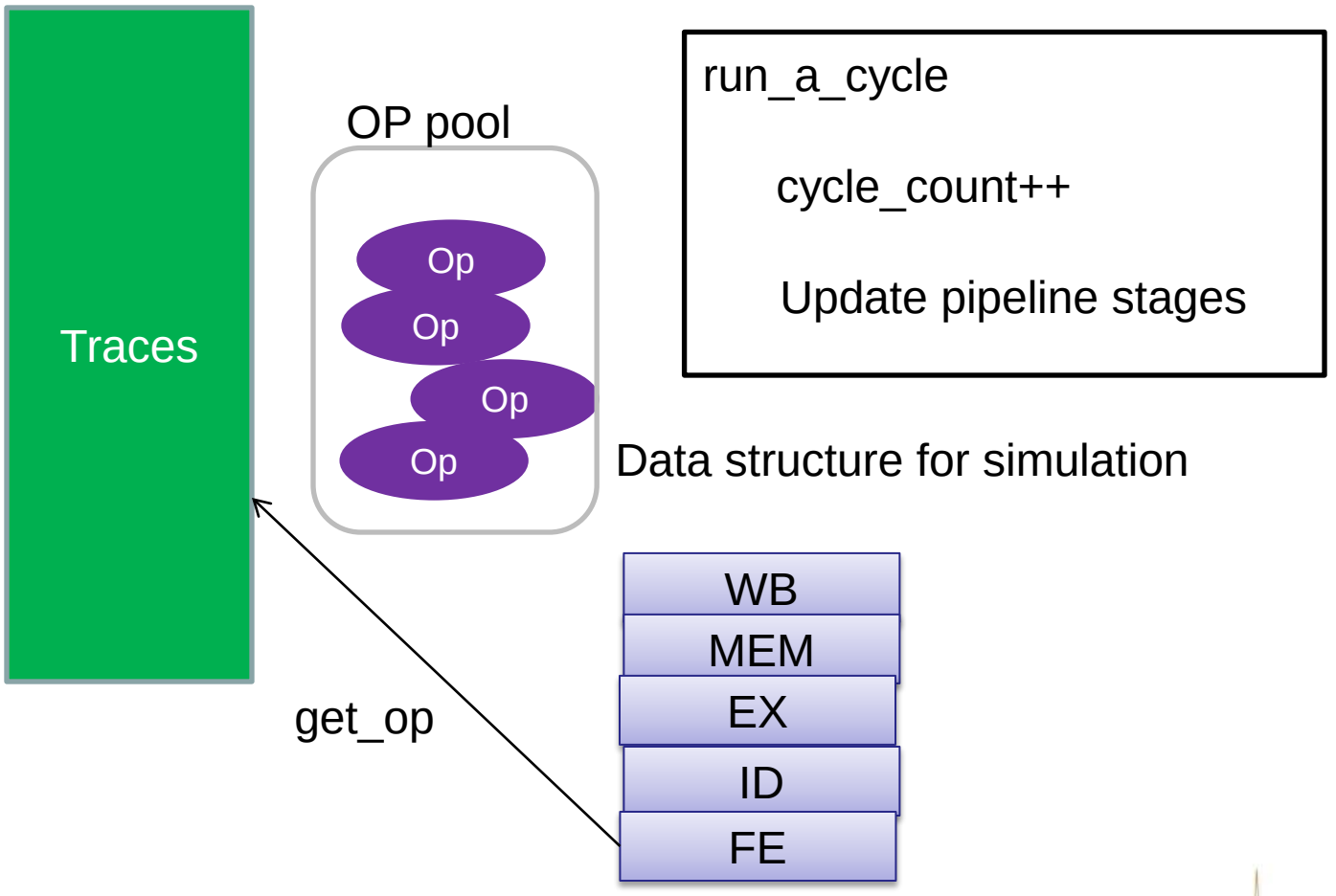
Broadcasting





Assignment #1

- Sim.cpp
- Op structure,
- Op latency
- Pipeline latch
- Get Op


Traces
OP pool
Op
Op
Op
Op
run_a_cycle
cycle_count++
Update pipeline stages
Data structure for simulation
get_op
WB
MEM
EX
ID
FE

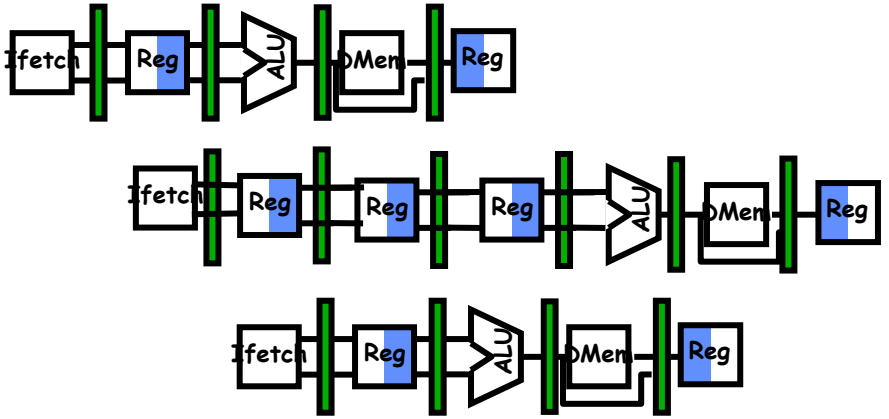


REGISTER RENAMEING

Dynamic scheduling

Instr. Order

```
add r1,r2,r3
sub r4,r1,r3
and r6,r2,r7
```



All sources are ready?
Why not execute them?



HW Register Renaming

- Give processor more registers than specified by the ISA
  - temporarily map ISA registers ("logical" or "architected" registers) to the *physical* registers to avoid overwrites
- Components:
  - mapping mechanism
  - physical registers
    - allocated vs. free registers
    - allocation/deallocation mechanism

Register Renaming

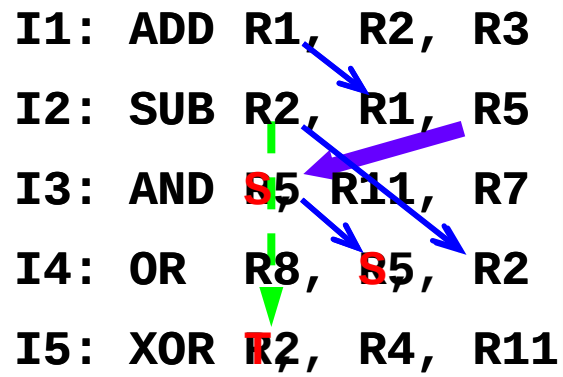
- Example
  - I3 can not exec before I2 because I3 will overwrite R5
  - I5 can not go before I2 because I2, when it goes, will overwrite R2 with a stale value

Program code

```
I1: ADD R1, R2, R3
I2: SUB R2, R1, R5
I3: AND R5, R11, R7
I4: OR  R8, R5, R2
I5: XOR R2, R4, R11
```

Register Renaming

- Solution:
  Let's give I3 temporary name/ location (e.g., S) for the value it produces.



- But I4 uses that value,
  so we must also change that to S…
- In fact, all uses of R5 from I3 to the next instruction that writes to R5 again must now be changed to S!
- We remove WAW deps in the same way: change R2 in I5 (and subsequent instrs) to T.

Register Renaming

- Implementation
  - Space for S, T, etc.
  - How do we know when to rename a register?
- Simple Solution
  - Do renaming for every instruction
  - Change the name of a register each time we decode an instruction that will write to it.
  - Remember what name we gave it ☺

Program code

```
I1: ADD R1, R2, R3
I2: SUB R2, R1, R5
I3: AND S, R11, R7
I4: OR  R8, S,  R2
I5: XOR T, R4, R11
```