

EECS 583 – Class 11

Instruction Scheduling

University of Michigan

February 13, 2023

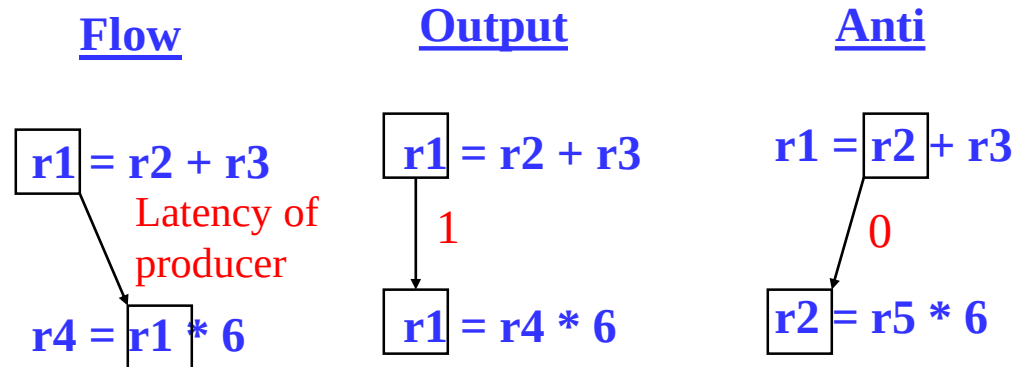
Announcements & Reading Material

- ❖ HW 2 – Due Wednesday at midnight!
 - » Talk to Aditya for last minute help
- ❖ Project discussion meetings
 - » Project proposal meeting signup next week – Signup on Google Calendar
 - » Each group meets 10 mins with Aditya and I
 - » Action items
 - Need to identify group members
 - Use piazza to recruit additional group members or express your availability
 - Think about general project areas that you want to work on
- ❖ Today's class
 - » “The Importance of Prepass Code Scheduling for Superscalar and Superpipelined Processors,” P. Chang et al., IEEE Transactions on Computers, 1995, pp. 353-370.
- ❖ Next class
 - » “Iterative Modulo Scheduling: An Algorithm for Software Pipelining Loops”, B. Rau, MICRO-27, 1994, pp. 63-74.

From Last Time: Data Dependences + Latencies

❖ Data dependences

- » If 2 operations access the same register, they are dependent
- » However, only keep dependences to most recent producer/consumer as other edges are redundant
- » Types of data dependences



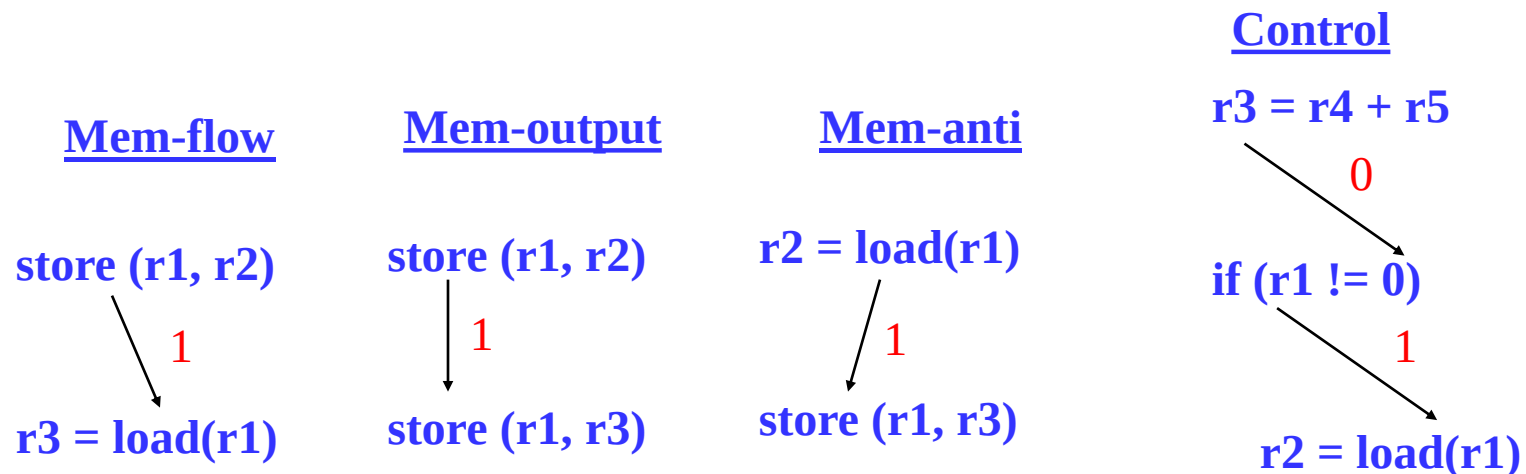
From Last Time: More Dependences + Latencies

❖ Memory dependences

- » Similar as register, but through memory
- » Memory dependences may be certain or maybe

❖ Control dependences

- » Branch determines whether an operation is executed or not
- » Operation must execute after/before a branch



Class Problem – Add Latencies to Dependence Edges

latencies

add: 1
cmpp: 1
load: 2
store: 1



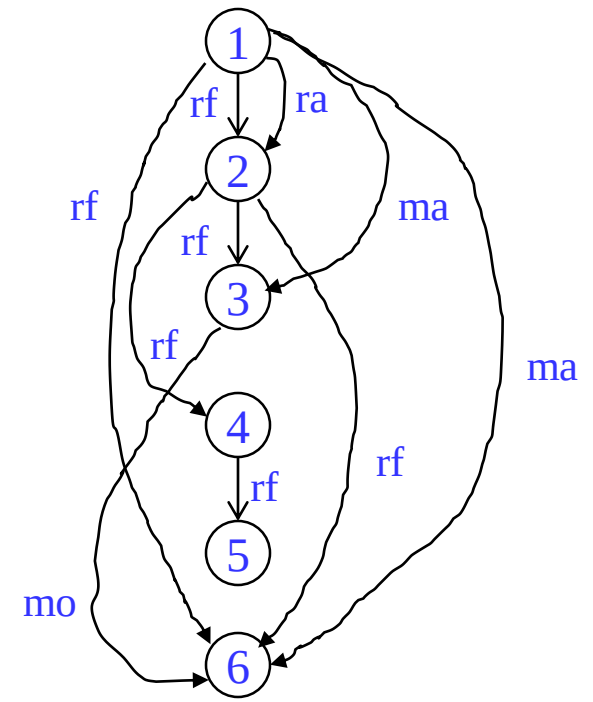
Example

1: r1 = load(r2)
2: r2 = r1 + r4
3: store (r4, r2)
4: p1 = cmpp (r2 < 0)
5: branch if p1 to BB3
6: store (r1, r2)

BB3:

Instructions 1-4 have control dependence to instruction 5

5→6 control dependence



Homework Problem 1

machine model

latencies

add: 1
mpy: 3
load: 2
store: 1

1. Draw dependence graph
2. Label edges with type and latencies

1. $r1 = \text{load}(r2)$
2. $r2 = r2 + 1$
3. $\text{store}(r8, r2)$
4. $r3 = \text{load}(r2)$
5. $r4 = r1 * r3$
6. $r5 = r5 + r4$
7. $r2 = r6 + 4$
8. $\text{store}(r2, r5)$

①

②

③

④

⑤

⑥

⑦

⑧

Homework Problem 1: Answer

machine model

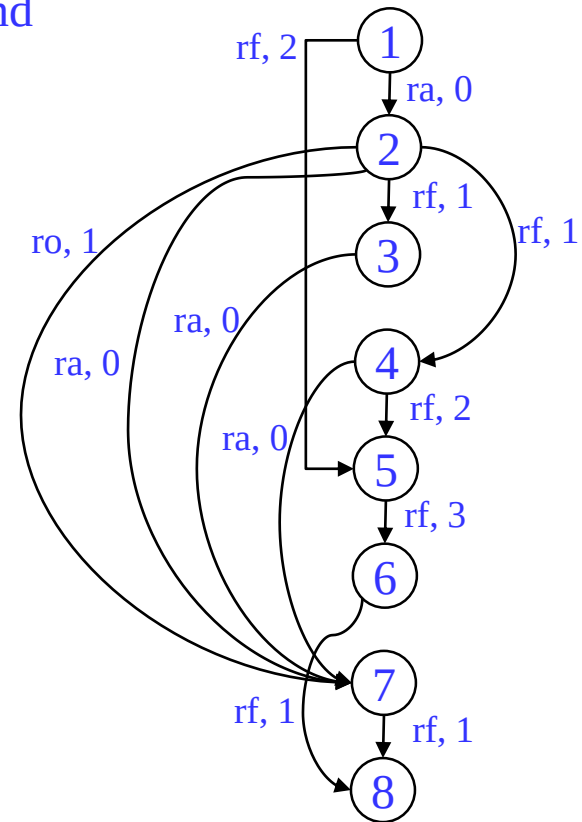
latencies

add: 1
mpy: 3
load: 2
store: 1

Store format (addr, data)

1. Draw dependence graph
2. Label edges with type and latencies

1. $r1 = \text{load}(r2)$
2. $r2 = r2 + 1$
3. $\text{store}(r8, r2)$
4. $r3 = \text{load}(r2)$
5. $r4 = r1 * r3$
6. $r5 = r5 + r4$
7. $r2 = r6 + 4$
8. $\text{store}(r2, r5)$

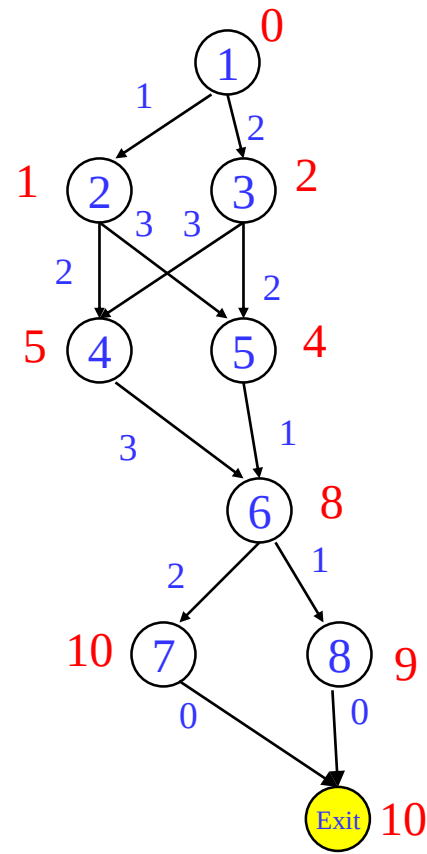


Memory deps all with latency =1: $1 \rightarrow 3$ (ma), $1 \rightarrow 8$ (ma),
 $3 \rightarrow 4$ (mf), $3 \rightarrow 8$ (mo), $4 \rightarrow 8$ (ma)

No control ⁶dependences

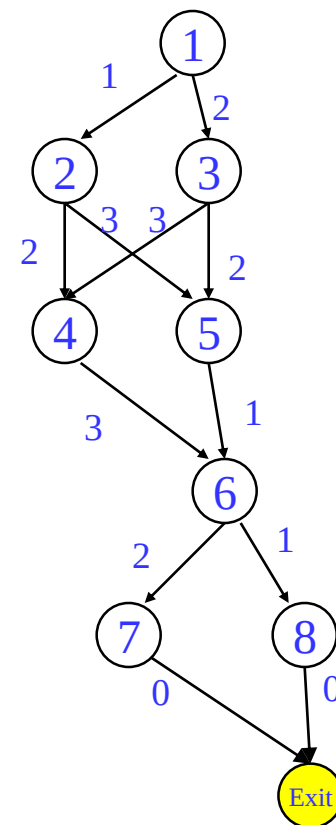
Dependence Graph Properties - Estart

- ❖ Estart = earliest start time, (as soon as possible - ASAP)
 - » Schedule length with infinite resources (dependence height)
 - » Estart = 0 if node has no predecessors
 - » $Estart = \text{MAX}(Estart(\text{pred}) + \text{latency})$ for each predecessor node
 - » Example



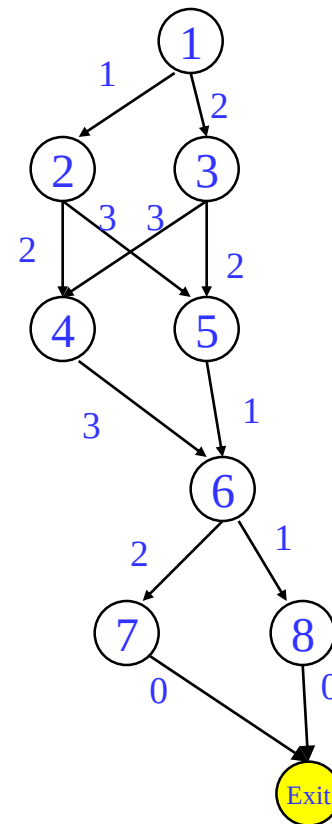
Lstart

- ❖ Lstart = latest start time, ALAP
 - » Latest time a node can be scheduled s.t. sched length not increased beyond infinite resource schedule length
 - » Lstart = Estart if node has no successors
 - » $Lstart = \min(Lstart(succ) - latency)$ for each successor node
 - » Example



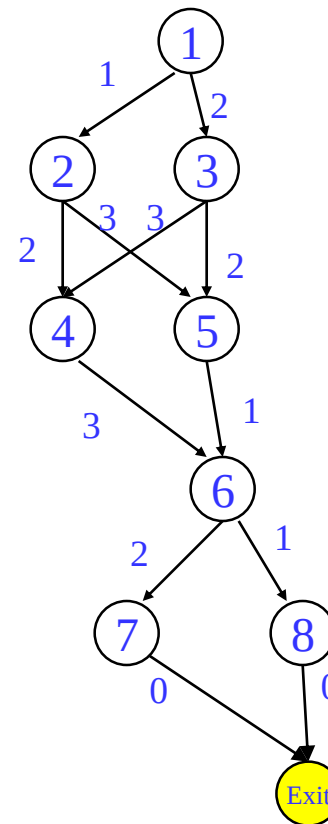
Slack

- ❖ Slack = measure of the scheduling freedom
 - » Slack = $L_{start} - E_{start}$ for each node
 - » Larger slack means more mobility
 - » Example

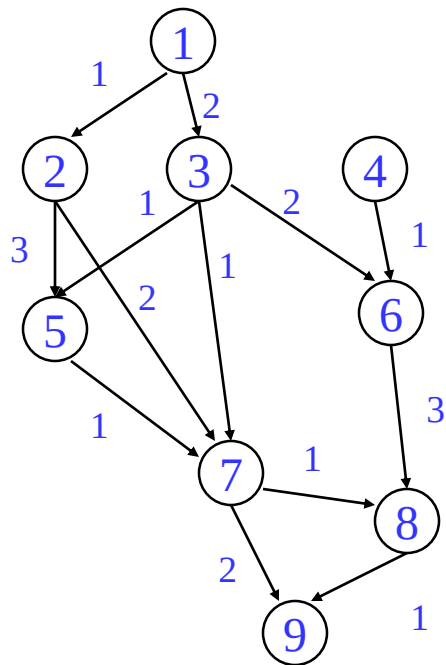


Critical Path

- ❖ Critical operations = Operations with slack = 0
 - » No mobility, cannot be delayed without extending the schedule length of the block
 - » Critical path = sequence of critical operations from node with no predecessors to exit node, can be multiple crit paths



Homework Problem 2



Node	Estart	Lstart	Slack
------	--------	--------	-------

1			
---	--	--	--

2			
---	--	--	--

3			
---	--	--	--

4			
---	--	--	--

5			
---	--	--	--

6			
---	--	--	--

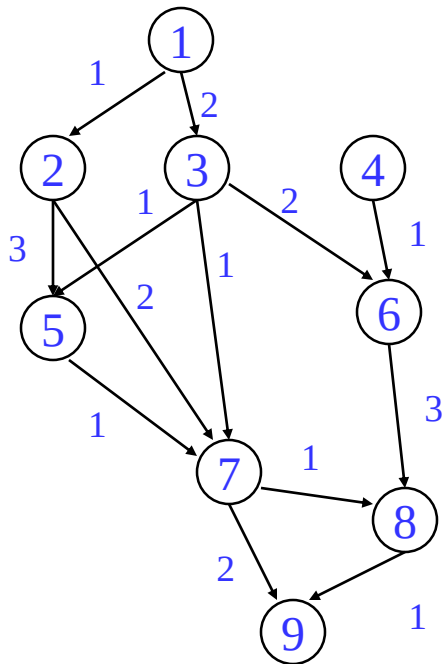
7			
---	--	--	--

8			
---	--	--	--

9			
---	--	--	--

Critical path(s) =

Homework Problem 2 - Answer



Node	Estart	Lstart	Slack
1	0	0	0
2	1	2	2
3	2	2	0
4	0	3	3
5	4	5	1
6	4	4	0
7	5	6	1
8	7	7	0
9	8	8	0

Critical path(s) = 1,3,6,8,9

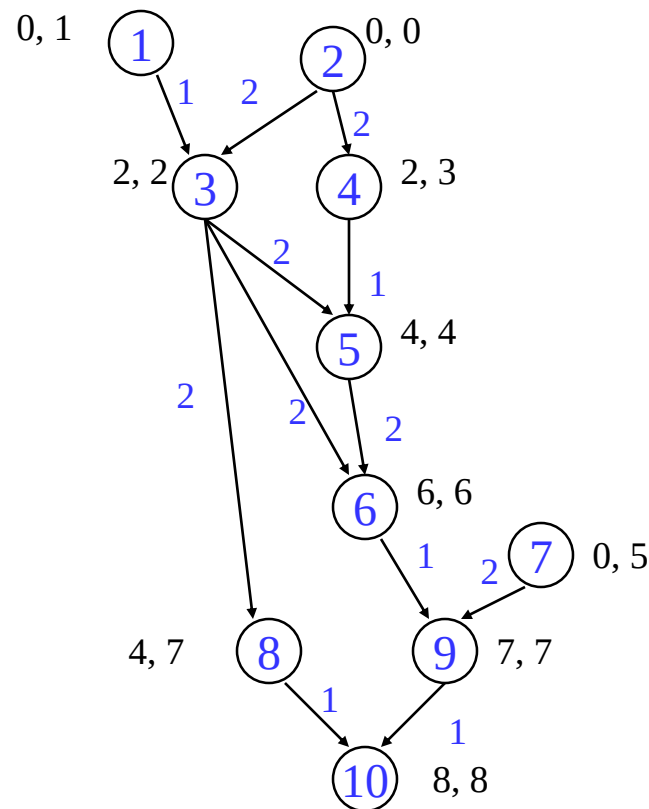
Operation Priority

- ❖ Priority – Need a mechanism to decide which ops to schedule first (when you have multiple choices)
- ❖ Common priority functions
 - » Height – Distance from exit node
 - Give priority to amount of work left to do
 - » Slackness – inversely proportional to slack
 - Give priority to ops on the critical path
 - » Register use – priority to nodes with more source operands and fewer destination operands
 - Reduces number of live registers
 - » Uncover – high priority to nodes with many children
 - Frees up more nodes
 - » Original order – when all else fails

Height-Based Priority

❖ Height-based is the most common

» $\text{priority}(\text{op}) = \text{MaxLstart} - \text{Lstart}(\text{op}) + 1$



op	priority
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

List Scheduling (aka Cycle Scheduler)

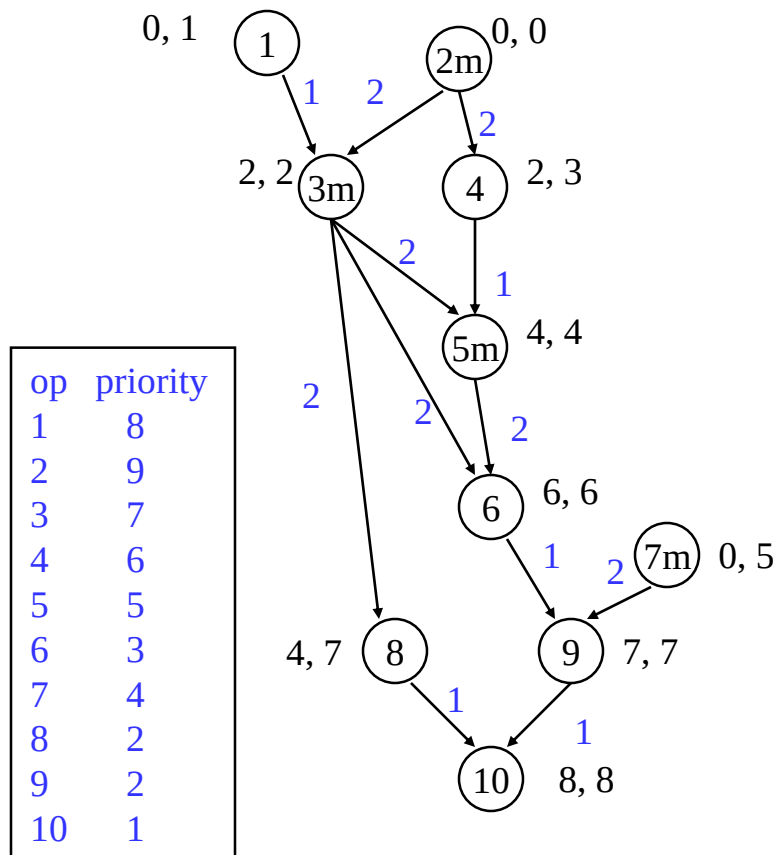
- ❖ Build dependence graph, calculate priority
- ❖ Add all ops to UNSCHEDULED set
- ❖ time = -1
- ❖ while (UNSCHEDULED is not empty)
 - » time++
 - » READY = UNSCHEDULED ops whose incoming dependences have been satisfied
 - » Sort READY using priority function
 - » For each op in READY (highest to lowest priority)
 - op can be scheduled at current time? (are the resources free?)
 - ◆ Yes, schedule it, op.issue_time = time
 - ↓ Mark resources busy in RU_map relative to issue time
 - ↓ Remove op from UNSCHEDULED/READY sets
 - ◆ No, continue

Cycle Scheduling Example

Processor: 2 issue, 1 memory port, 1 ALU

Memory port = 2 cycles, pipelined

ALU = 1 cycle



RU_map			Schedule	
time	ALU	MEM	time	Instructions
0			0	
1			1	
2			2	
3			3	
4			4	
5			5	
6			6	
7			7	
8			8	
9			9	

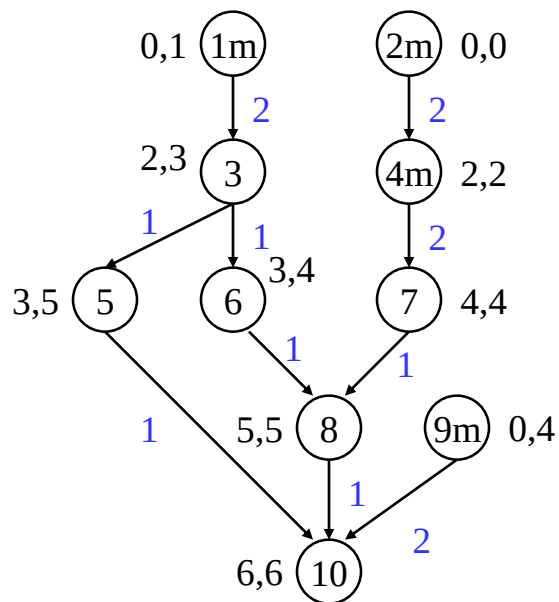
Time =
Ready =

Homework Problem 3

Processor: 2 issue, 1 memory port, 1 ALU

Memory port = 2 cycles, pipelined

ALU = 1 cycle



RU_map			Schedule	
time	ALU	MEM	time	Instructions
0			0	
1			1	
2			2	
3			3	
4			4	
5			5	
6			6	
7			7	
8			8	
9			9	

Time =

Ready =

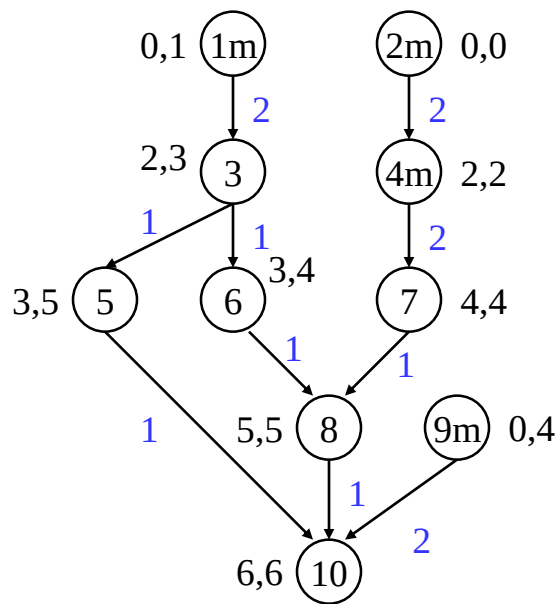
1. Calculate height-based priorities
2. Schedule using cycle scheduler

Homework Problem 3 – Answer

Processor: 2 issue, 1 memory port, 1 ALU

Memory port = 2 cycles, pipelined

ALU = 1 cycle



Op	priority
1	6
2	7
3	4
4	5
5	2
6	3
7	3
8	2
9	3
10	1

RU_map

time	ALU	MEM
0		X
1		X
2		X
3	X	X
4	X	
5	X	
6	X	
7	X	
8	X	

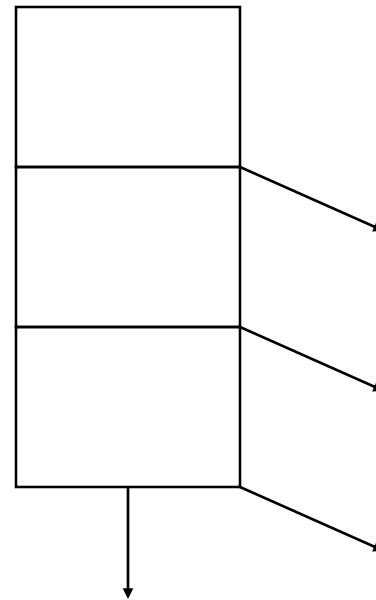
Schedule

Time	Instructions
0	2
1	1
2	4
3	3, 9
4	6
5	7
6	5
7	8
8	10

1. Calculate height-based priorities
2. Schedule using Operation scheduler

Generalize Beyond a Basic Block

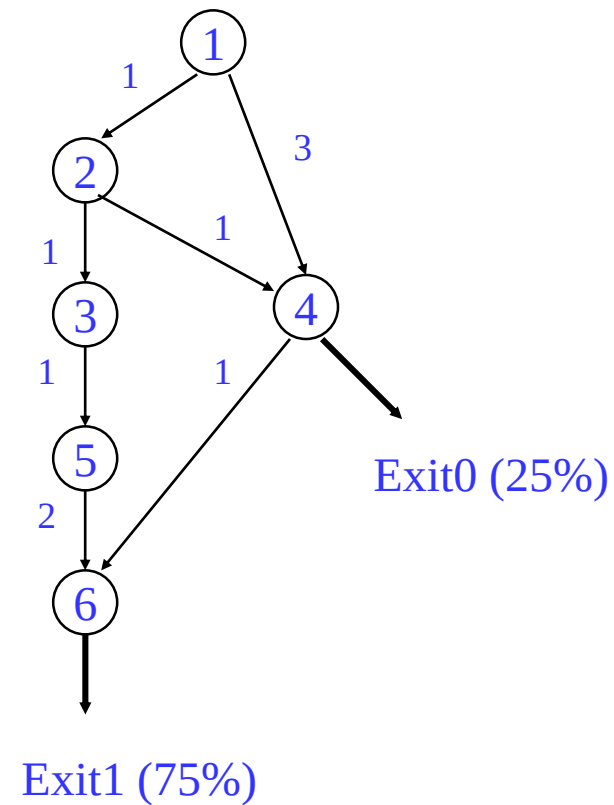
- ❖ Superblock
 - » Single entry
 - » Multiple exits (side exits)
 - » No side entries
- ❖ Schedule just like a BB
 - » Priority calculations needs change
 - » Dealing with control deps



Lstart in a Superblock

- ❖ Not a single Lstart any more
 - » 1 per exit branch (Lstart is a vector!)
 - » Exit branches have probabilities

op	Estart	Lstart0	Lstart1
1			
2			
3			
4			
5			
6			



Operation Priority in a Superblock

❖ Priority – Dependence height and speculative yield

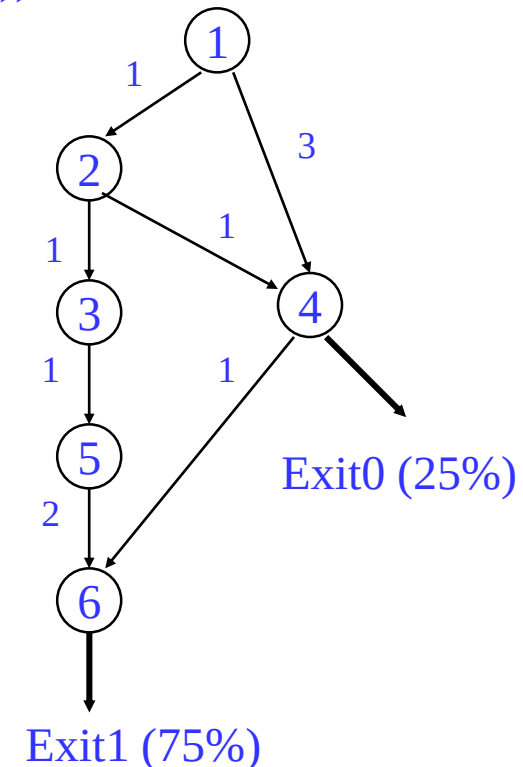
» Height from op to exit * probability of exit

» Sum up across all exits in the superblock

$$\text{Priority}(\text{op}) = \text{SUM}(\text{Probi} * (\text{MAX_Lstart} - \text{Lstarti}(\text{op}) + 1))$$

valid late times for op

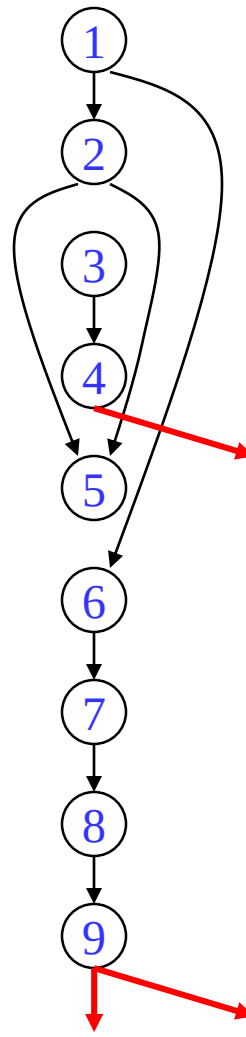
op	Lstart0	Lstart1	Priority
1			
2			
3			
4			
5			
6			



Dependences in a Superblock

Superblock

```
1: r1 = r2 + r3
2: r4 = load(r1)
3: p1 = cmpp(r3 == 0)
4: branch p1 Exit1
5: store (r4, -1)
6: r2 = r2 - 4
7: r5 = load(r2)
8: p2 = cmpp(r5 > 9)
9: branch p2 Exit2
```



* Data dependences shown, all are reg flow except $1 \rightarrow 6$ is reg anti

* Dependences define precedence ordering of operations to ensure correct execution semantics

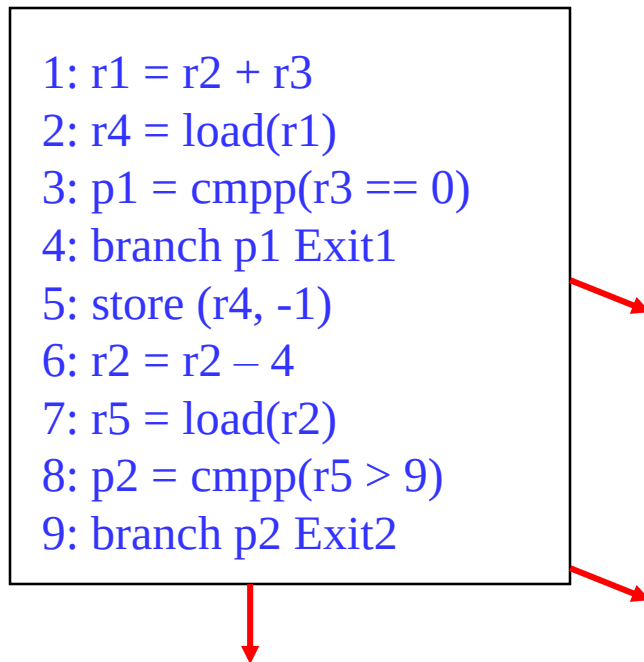
* What about control dependences?

* Control dependences define precedence of ops with respect to branches

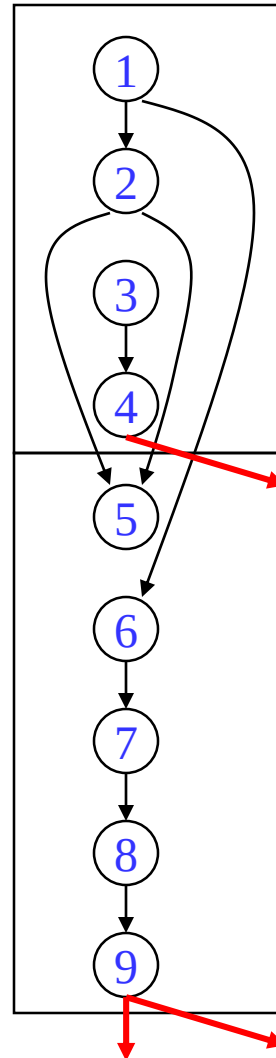
Note: Control flow in red bold

Conservative Approach to Control Dependences

Superblock



Note: Control flow in red bold



- * Make branches barriers, nothing moves above or below branches

- * Schedule each BB in SB separately

- * Sequential schedules

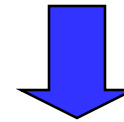
- * Whole purpose of a superblock is lost

- * Need a better solution!

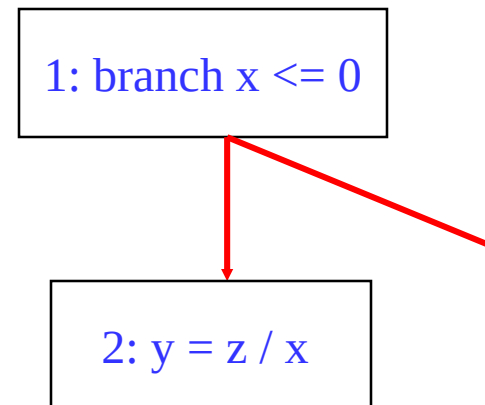
Upward Code Motion Across Branches

- ❖ Restriction 1a (register op)
 - » The destination of op is not in `liveout(br)`
 - » Wrongly kill a live value
- ❖ Restriction 1b (memory op)
 - » Op does not modify the memory
 - » Actually live memory is what matters, but that is often too hard to determine
- ❖ Restriction 2
 - » Op must not cause an exception that may terminate the program execution when br is taken
 - » Op is executed more often than it is supposed to (speculated)
 - » Page fault or cache miss are ok
- ❖ Insert control dep when either restriction is violated

...
if (x > 0)
 y = z / x
...



control flow graph



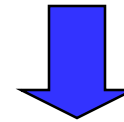
Downward Code Motion Across Branches

- ❖ Restriction 1 (liveness)
 - » If no compensation code
 - Same restriction as before, destination of op is not liveout
 - » Else, no restrictions
 - Duplicate operation along both directions of branch if destination is liveout
- ❖ Restriction 2 (speculation)
 - » Not applicable, downward motion is not speculation
- ❖ Again, insert control dep when the restrictions are violated
- ❖ Part of the philosophy of superblocks is no compensation code insertion hence R1 is enforced!

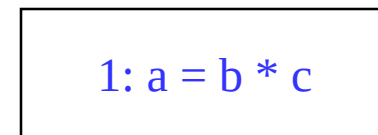
...
a = b * c
if (x > 0)

else

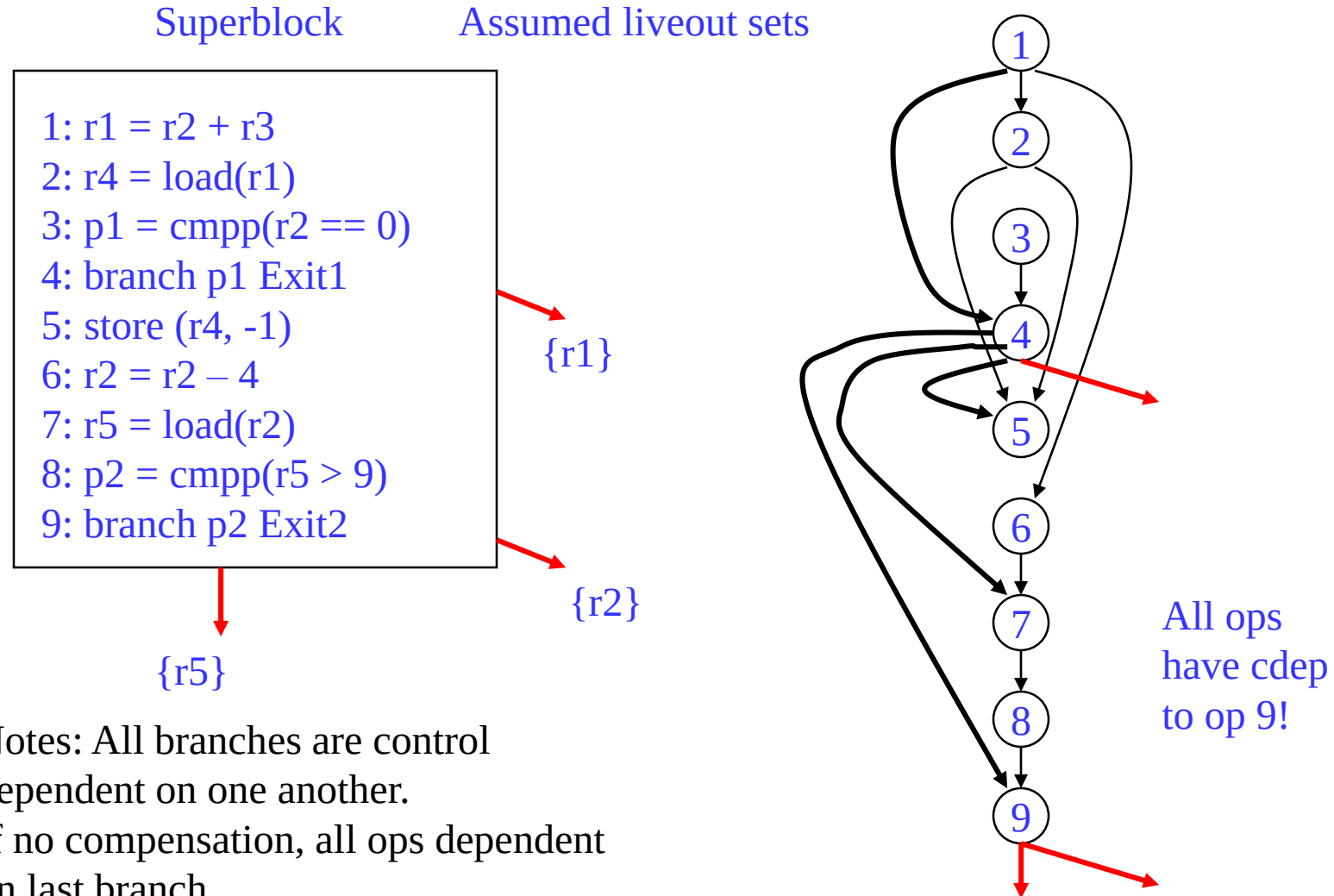
...



control flow graph



Add Control Dependences to a Superblock



Notes: All branches are control dependent on one another.
If no compensation, all ops dependent on last branch

To Be Continued