EECS 583 – Class 14
Modulo Scheduling Reloaded

University of Michigan

October 31, 2018

Announcements + Reading Material

- Midterm exam – Wednesday Nov 14 in class (10:40-12:20)
  - 3 practice exams posted on course website – our exam will be a little shorter than prior exams
  - Covers lecture material up through register allocation
  - Go through example/class/homework problems from lectures
  - No LLVM coding
- Today's class reading
  - "Code Generation Schema for Modulo Scheduled Loops", B. Rau, M. Schlansker, and P. Tirumalai, MICRO-25, Dec. 1992.
- Next class reading
  - "Register Allocation and Spilling Via Graph Coloring," G. Chaitin, Proc. 1982 SIGPLAN Symposium on Compiler Construction, 1982.

Modulo Scheduling Process

- Use list scheduling but we need a few twists
  - II is predetermined – starts at MII, then is incremented
  - Cyclic dependences complicate matters
    - Estart/Priority/etc.
    - Consumer scheduled before producer is considered
      - There is a window where something can be scheduled!
  - Guarantee the repeating pattern
- 2 constraints enforced on the schedule
  - Each iteration begin exactly II cycles after the previous one
  - Each time an operation is scheduled in 1 iteration, it is tentatively scheduled in subsequent iterations at intervals of II
    - MRT used for this

Priority Function

Height-based priority worked well for acyclic scheduling, makes sense that it will work for loops as well

Acyclic:

$$Height(X) = \begin{cases} 0, \text{ if X has no successors} \\ \underset{\text{for all Y = succ(X)}}{MAX} ((Height(Y) + Delay(X,Y)), \text{ otherwise} \end{cases}$$

Cyclic:

$$HeightR(X) = \begin{cases} 0, \text{ if X has no successors} \\ \underset{\text{for all Y = succ(X)}}{MAX} ((HeightR(Y) + EffDelay(X,Y)), \text{ otherwise} \end{cases}$$

$$EffDelay(X,Y) = Delay(X,Y) - II*Distance(X,Y)$$

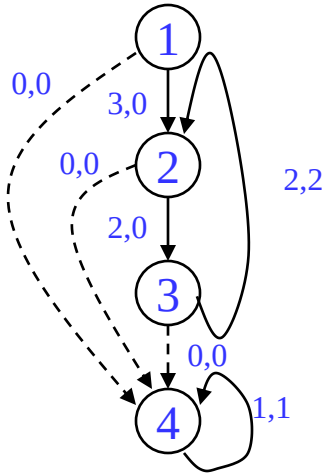
Calculating Height

1. Insert pseudo edges from all nodes to branch with latency = 0, distance = 0 (dotted edges)
2. Compute II, For this example assume II = 2
3. HeightR(4) =

4. HeightR(3) =

5. HeightR(2) =

6. HeightR(1)



The Scheduling Window

With cyclic scheduling, not all the predecessors may be scheduled, so a more flexible earliest schedule time is:

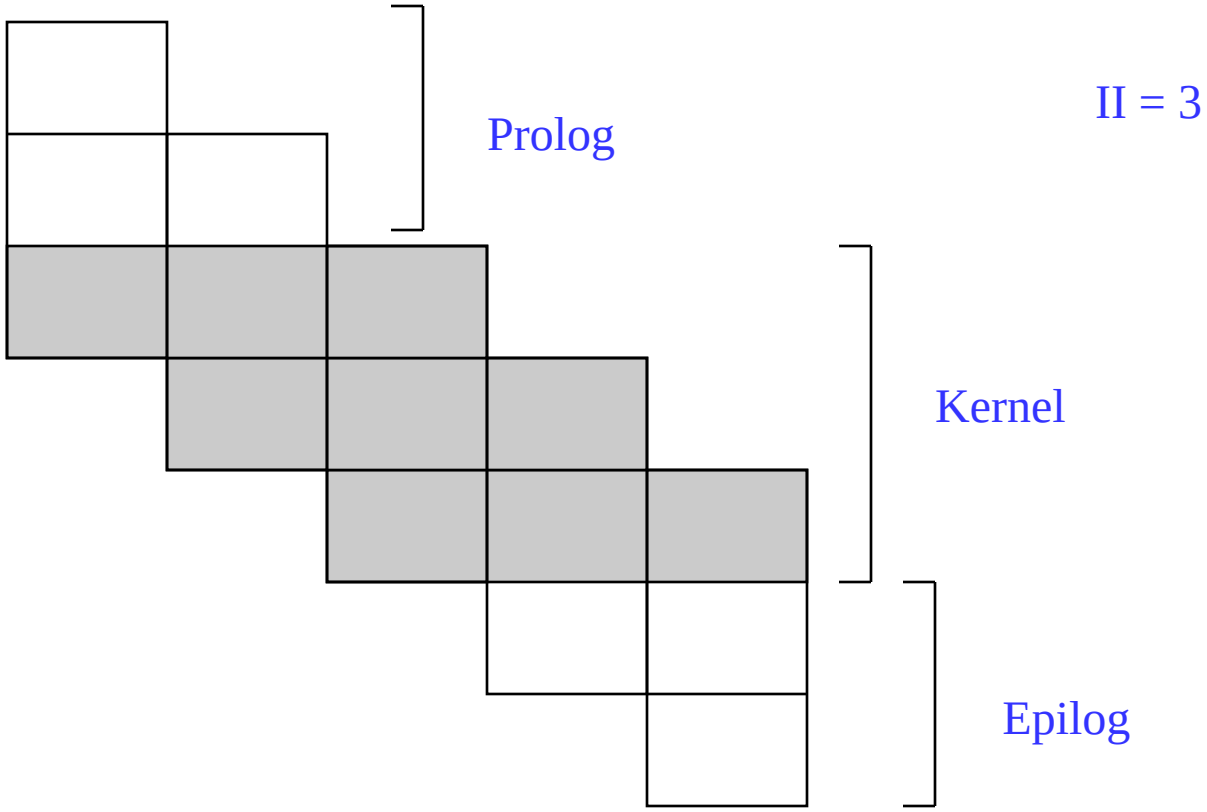
$$E(Y) = \max_{\text{for all } X = \text{pred}(Y)} \begin{cases} 0, \text{ if X is not scheduled} \\ \text{MAX } (0, \text{SchedTime}(X) + \text{EffDelay}(X,Y)), \text{ otherwise} \end{cases}$$

where $\text{EffDelay}(X,Y) = \text{Delay}(X,Y) - II * \text{Distance}(X,Y)$

Every II cycles a new loop iteration will be initialized, thus every II cycles the pattern will repeat. Thus, you only have to look in a window of size II, if the operation cannot be scheduled there, then it cannot be scheduled.

$$\text{Latest schedule time}(Y) = L(Y) = E(Y) + II - 1$$

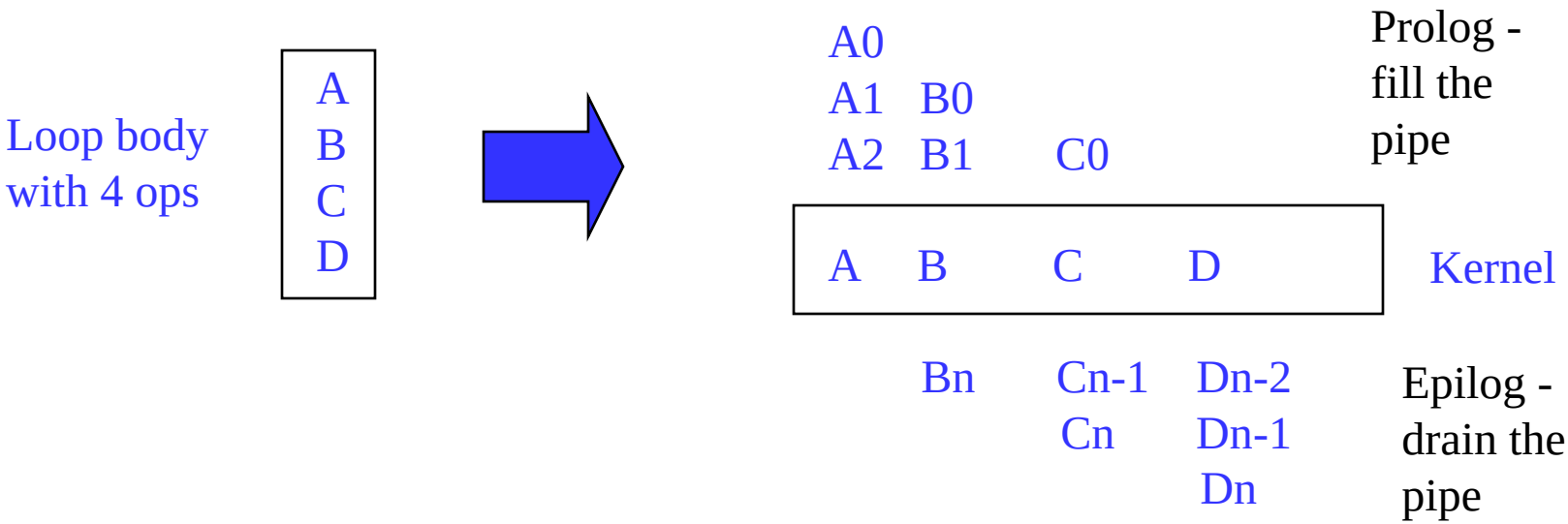
Loop Prolog and Epilog



Only the kernel involves executing full width of operations

Prolog and epilog execute a subset (ramp-up and ramp-down)

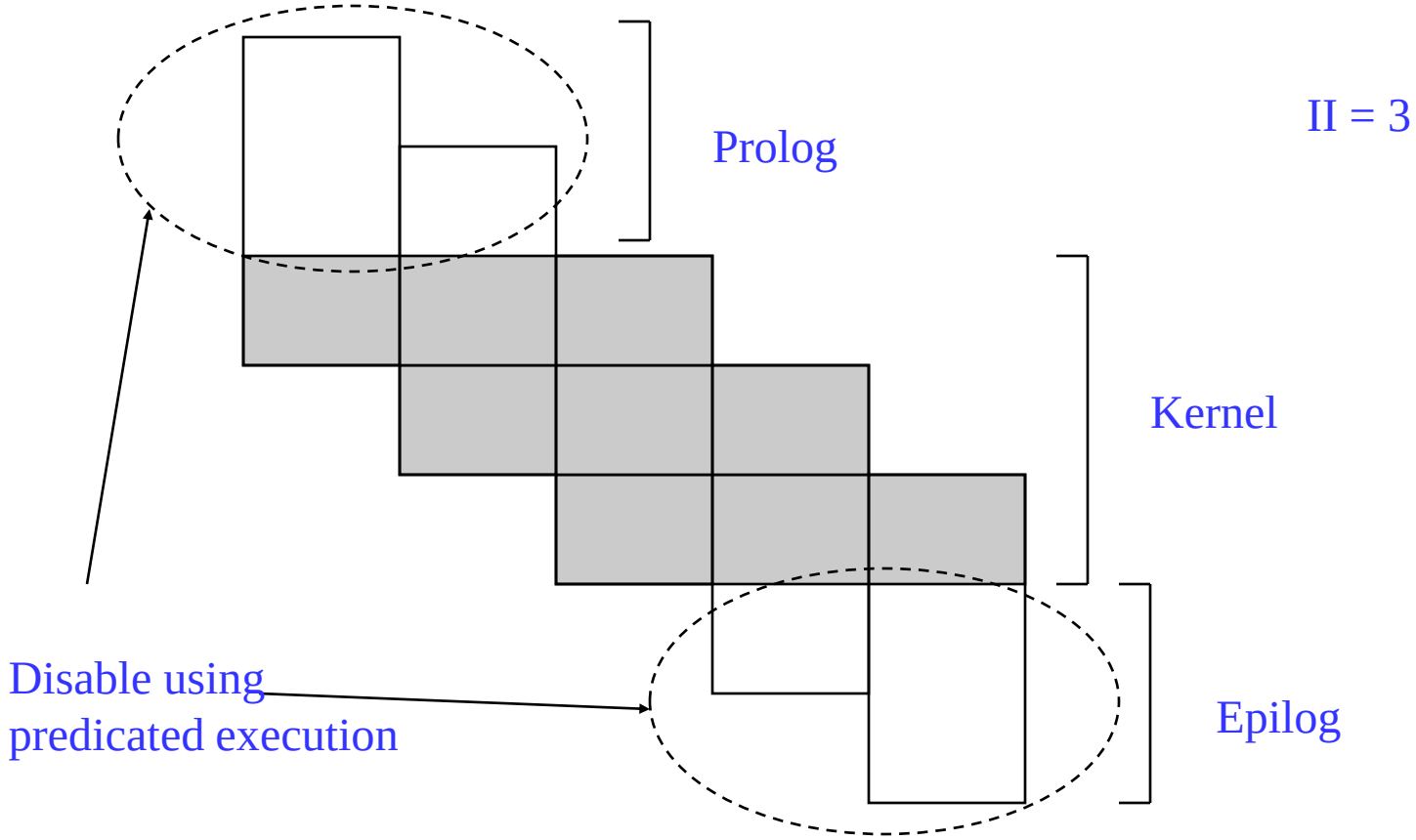
Separate Code for Prolog and Epilog



Generate special code before the loop (preheader) to fill the pipe and special code after the loop to drain the pipe.

Peel off II-1 iterations for the prolog. Complete II-1 iterations in epilog

Removing Prolog/Epilog

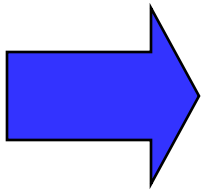


Execute loop kernel on every iteration, but for prolog and epilog selectively disable the appropriate operations to fill/drain the pipeline

Kernel-only Code Using Rotating Predicates

A0
A1 B0
A2 B1 C0

A B C D



A if P[0] B if P[1] C if P[2] D if P[3]

Bn Cn-1 Dn-2
Cn Dn-1
Dn

P referred to as the staging predicate

P[0]	P[1]	P[2]	P[3]
1	0	0	0
1	1	0	0
1	1	1	0
1	1	1	1
…			
0	1	1	1
0	0	1	1
0	0	0	1

A	-	-	-
A	B	-	-
A	B	C	-
A	B	C	D
…			
-	B	C	D
-	-	C	D
-	-	-	D

Modulo Scheduling Architectural Support

- Loop requiring N iterations
  - Will take N + (S – 1) where S is the number of stages
- 2 special registers created
  - LC: loop counter (holds N)
  - ESC: epilog stage counter (holds S)
- Software pipeline branch operations
  - Initialize LC = N, ESC = S in loop preheader
  - All rotating predicates are cleared
  - BRF.B.B.F
    - While LC > 0, decrement LC and RRB, P[0] = 1, branch to top of loop
      - This occurs for prolog and kernel
    - If LC = 0, then while ESC > 0, decrement RRB and write a 0 into P[0], and branch to the top of the loop
      - This occurs for the epilog

Execution History With LC/ESC

LC = 3, ESC = 3 /* Remember 0 relative!! */
Clear all rotating predicates
P[0] = 1

```
A if P[0];   B if P[1];   C if P[2];  D if P[3];  P[0] = BRF.B.B.F;
```

LC	ESC	P[0]	P[1]	P[2]	P[3]				
3	3	1	0	0	0	A			
2	3	1	1	0	0	A	B		
1	3	1	1	1	0	A	B	C	
0	3	1	1	1	1	A	B	C	D
0	2	0	1	1	1	-	B	C	D
0	1	0	0	1	1	-	-	C	D
0	0	0	0	0	1	-	-	-	D

4 iterations, 4 stages, II = 1, Note 4 + 4 –1 iterations of kernel executed

Implementing Modulo Scheduling - Driver

- compute MII
- II = MII
- budget = BUDGET_RATIO * number of ops
- while (schedule is not found) do
  - iterative_schedule(II, budget)
  - II++

- Budget_ratio is a measure of the amount of backtracking that can be performed before giving up and trying a higher II

Modulo Scheduling – Iterative Scheduler

- iterative_schedule(II, budget)
  - compute op priorities
  - while (there are unscheduled ops and budget > 0) do
    - op = unscheduled op with the highest priority
    - min = early time for op (E(Y))
    - max = min + II – 1
    - t = find_slot(op, min, max)
    - schedule op at time t
      - /* Backtracking phase – undo previous scheduling decisions */
      - Unschedule all previously scheduled ops that conflict with op
    - budget--

Modulo Scheduling – Find_slot

- find_slot(op, min, max)
  - /* Successively try each time in the range */
  - for (t = min to max) do
    - if (op has no resource conflicts in MRT at t)
      - return t
  - /* Op cannot be scheduled in its specified range */
  - /* So schedule this op and displace all conflicting ops */
  - if (op has never been scheduled or min > previous scheduled time of op)
    - return min
  - else
    - return MIN(1 + prev scheduled time of op, max)

Recall: The Scheduling Window

With cyclic scheduling, not all the predecessors may be scheduled, so a more flexible earliest schedule time is:

$$E(Y) = \max_{\text{for all } X = \text{pred}(Y)} \begin{cases} 0, \text{ if } X \text{ is not scheduled} \\ \max(0, \text{SchedTime}(X) + \text{EffDelay}(X,Y)), \text{ otherwise} \end{cases}$$

where $\text{EffDelay}(X,Y) = \text{Delay}(X,Y) - II * \text{Distance}(X,Y)$

Every II cycles a new loop iteration will be initialized, thus every II cycles the pattern will repeat. Thus, you only have to look in a window of size II, if the operation cannot be scheduled there, then it cannot be scheduled.

$$\text{Latest schedule time}(Y) = L(Y) = E(Y) + II - 1$$

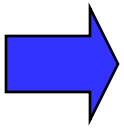
Modulo Scheduling Example

resources: 4 issue, 2 alu, 1 mem, 1 br
latencies: add=1, mpy=3, ld = 2, st = 1, br = 1

```
for (j=0; j<100; j++)
   b[j] = a[j] * 26
```

Step1: Compute to loop into form that uses LC

```
Loop: 1: r3 = load(r1)
      2: r4 = r3 * 26
      3: store (r2, r4)
      4: r1 = r1 + 4
      5: r2 = r2 + 4
      6: p1 = cmpp (r1 < r9)
      7: brct p1 Loop
```



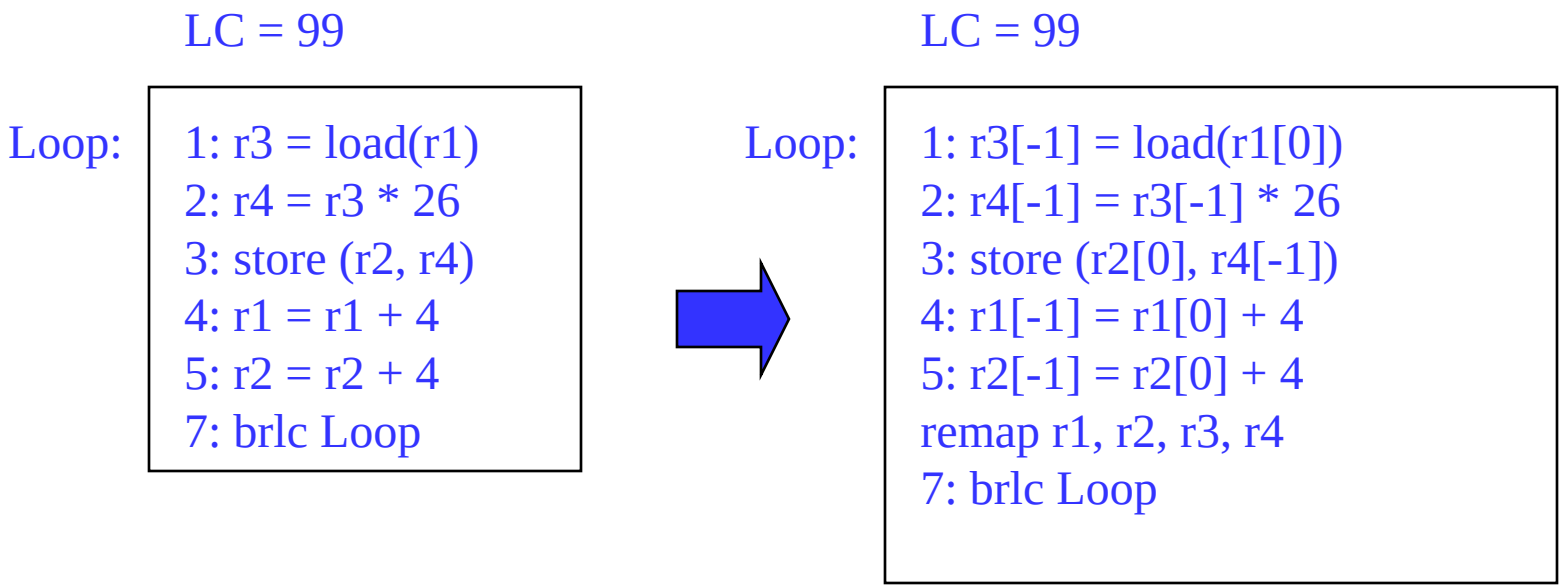
LC = 99

```
Loop: 1: r3 = load(r1)
      2: r4 = r3 * 26
      3: store (r2, r4)
      4: r1 = r1 + 4
      5: r2 = r2 + 4
      7: brlc Loop
```

Example – Step 2

resources: 4 issue, 2 alu, 1 mem, 1 br
latencies: add=1, mpy=3, ld = 2, st = 1, br = 1

Step 2: DSA convert



Example – Step 3

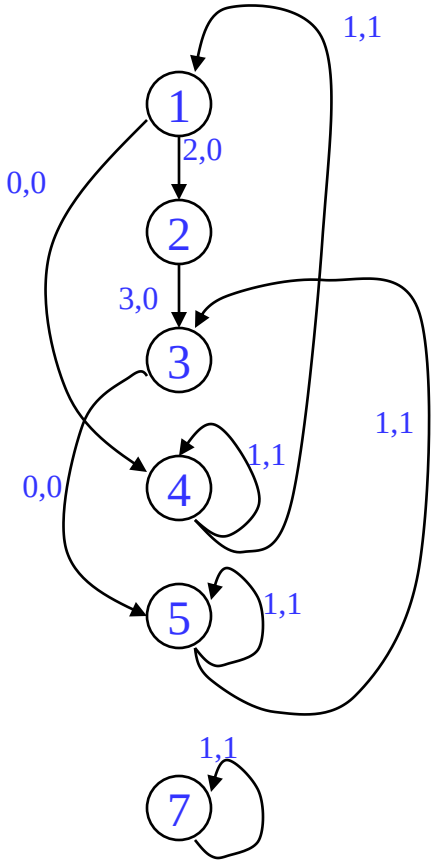
resources: 4 issue, 2 alu, 1 mem, 1 br
latencies: add=1, mpy=3, ld = 2, st = 1, br = 1

Step3: Draw dependence graph
Calculate MII

LC = 99

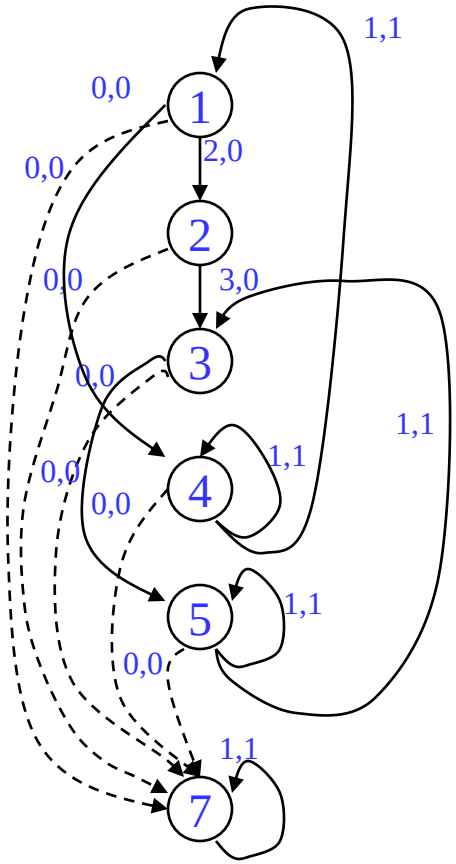
Loop:

```
1: r3[-1] = load(r1[0])
2: r4[-1] = r3[-1] * 26
3: store (r2[0], r4[-1])
4: r1[-1] = r1[0] + 4
5: r2[-1] = r2[0] + 4
remap r1, r2, r3, r4
7: brlc Loop
```



RecMII = 1
RESMII = 2
MII = 2

Example – Step 4



Step 4 – Calculate priorities (MAX height to pseudo stop node)

Iter1	Iter2
1: H = 5	1: H = 5
2: H = 3	2: H = 3
3: H = 0	3: H = 0
4: H = 0	4: H = 4
5: H = 0	5: H = 0
7: H = 0	7: H = 0

Example – Step 5

resources: 4 issue, 2 alu, 1 mem, 1 br
latencies: add=1, mpy=3, ld = 2, st = 1, br = 1

Schedule brlc at time II - 1

LC = 99

Loop:

```
1: r3[-1] = load(r1[0])
2: r4[-1] = r3[-1] * 26
3: store (r2[0], r4[-1])
4: r1[-1] = r1[0] + 4
5: r2[-1] = r2[0] + 4
remap r1, r2, r3, r4
7: brlc Loop
```

Rolled Schedule

0	
1	7

Unrolled Schedule

0	
1	
2	
3	
4	
5	
6	

MRT

	alu0	alu1	mem	br
0				
1				X

Example – Step 6

Step6: Schedule the highest priority op

Op1: E = 0, L = 1
Place at time 0 (0 % 2)

LC = 99

Loop:

```
1: r3[-1] = load(r1[0])
2: r4[-1] = r3[-1] * 26
3: store (r2[0], r4[-1])
4: r1[-1] = r1[0] + 4
5: r2[-1] = r2[0] + 4
remap r1, r2, r3, r4
7: brlc Loop
```

Rolled Schedule

0	1
1	7

Unrolled Schedule

0	1
1	
2	
3	
4	
5	
6	

MRT

	alu0	alu1	mem	br
0			X	
1				X

Example – Step 7

Step7: Schedule the highest priority op

Op4: E = 0, L = 1
Place at time 0 (0 % 2)

LC = 99

Loop:

```
1: r3[-1] = load(r1[0])
2: r4[-1] = r3[-1] * 26
3: store (r2[0], r4[-1])
4: r1[-1] = r1[0] + 4
5: r2[-1] = r2[0] + 4
remap r1, r2, r3, r4
7: brlc Loop
```

Rolled Schedule

0	1 4
1	7

Unrolled Schedule

0	1 4
1	
2	
3	
4	
5	
6	

MRT

	alu0	alu1	mem	br
0	X		X	
1				X

Example – Step 8

Step8: Schedule the highest priority op

Op2: E = 2, L = 3
Place at time 2 (2 % 2)

LC = 99

Loop:

```
1: r3[-1] = load(r1[0])
2: r4[-1] = r3[-1] * 26
3: store (r2[0], r4[-1])
4: r1[-1] = r1[0] + 4
5: r2[-1] = r2[0] + 4
remap r1, r2, r3, r4
7: brlc Loop
```

Rolled Schedule

0	1 4 2
1	7

Unrolled Schedule

0	1 4
1	
2	2
3	
4	
5	
6	

MRT

	alu0	alu1	mem	br
0	X	X	X	
1				X

Example – Step 9

Step9: Schedule the highest priority op

Op3: E = 5, L = 6
Place at time 5 (5 % 2)

LC = 99

Loop:

```
1: r3[-1] = load(r1[0])
2: r4[-1] = r3[-1] * 26
3: store (r2[0], r4[-1])
4: r1[-1] = r1[0] + 4
5: r2[-1] = r2[0] + 4
remap r1, r2, r3, r4
7: brlc Loop
```

Rolled Schedule

0	1 2 4
1	7 3

Unrolled Schedule

0	1 4
1	
2	2
3	
4	
5	3
6	

MRT

	alu0	alu1	mem	br
0	X	X	X	
1			X	X

Example – Step 10

Step10: Schedule the highest priority op

Op5: E = 5, L = 6
Place at time 5 (5 % 2)

LC = 99

Loop:

```
1: r3[-1] = load(r1[0])
2: r4[-1] = r3[-1] * 26
3: store (r2[0], r4[-1])
4: r1[-1] = r1[0] + 4
5: r2[-1] = r2[0] + 4
remap r1, r2, r3, r4
7: brlc Loop
```

Rolled Schedule

0	1 2 4
1	7 3 5

Unrolled Schedule

0	1 4
1	
2	2
3	
4	
5	3 5
6	

MRT

	alu0	alu1	mem	br
0	X	X	X	
1	X		X	X

Example – Step 11

Step11: calculate ESC, SC = ceiling(max unrolled sched length / ii)
unrolled sched time of branch = rolled sched time of br + (ii*esc)

SC = 6 / 2 = 3, ESC = SC – 1
time of br = 1 + 2*2 = 5

LC = 99

Loop:

```
1: r3[-1] = load(r1[0])
2: r4[-1] = r3[-1] * 26
3: store (r2[0], r4[-1])
4: r1[-1] = r1[0] + 4
5: r2[-1] = r2[0] + 4
remap r1, r2, r3, r4
7: brlc Loop
```

Rolled Schedule

0	1 2 4
1	7 3 5

Unrolled Schedule

0	1 4
1	
2	2
3	
4	
5	3 5 7
6	

MRT

	alu0	alu1	mem	br
0	X	X	X	
1	X		X	X

Example – Step 12

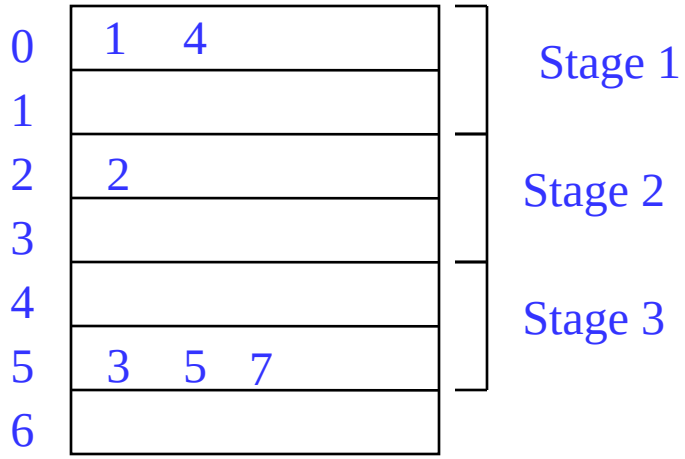
Finishing touches - Sort ops, initialize ESC, insert BRF and staging predicate, initialize staging predicate outside loop

LC = 99
ESC = 2
p1[0] = 1

Staging predicate, each successive stage increment the index of the staging predicate by 1, stage 1 gets px[0]

```
Loop:  1: r3[-1] = load(r1[0]) if p1[0]
       2: r4[-1] = r3[-1] * 26 if p1[1]
       4: r1[-1] = r1[0] + 4 if p1[0]
       3: store (r2[0], r4[-1]) if p1[2]
       5: r2[-1] = r2[0] + 4 if p1[2]
       7: brlc Loop if p1[2]
```

Unrolled Schedule



Example – Dynamic Execution of the Code

LC = 99
ESC = 2
p1[0] = 1

Loop:
1: r3[-1] = load(r1[0]) if p1[0]
2: r4[-1] = r3[-1] * 26 if p1[1]
4: r1[-1] = r1[0] + 4 if p1[0]
3: store (r2[0], r4[-1]) if p1[2]
5: r2[-1] = r2[0] + 4 if p1[2]
7: brlc Loop if p1[2]

Total time = II(num_iteration + num_stages – 1)
= 2(100 + 3 – 1) = 204 cycles

time: ops executed

0: 1, 4
1:

2: 1,2,4
3:

4: 1,2,4
5: 3,5,7

6: 1,2,4
7: 3,5,7

…

198: 1,2,4
199: 3,5,7

200: 2
201: 3,5,7

202: -
203 3,5,7

Class Problem

latencies: add=1, mpy=3, ld = 2, st = 1, br = 1

```
for (j=0; j<100; j++)
   b[j] = a[j] * 26
```

LC = 99

```
Loop:  1: r3 = load(r1)
       2: r4 = r3 * 26
       3: store (r2, r4)
       4: r1 = r1 + 4
       5: r2 = r2 + 4
       7: brlc Loop
```

How many resources of each type are required to achieve an II=1 schedule?

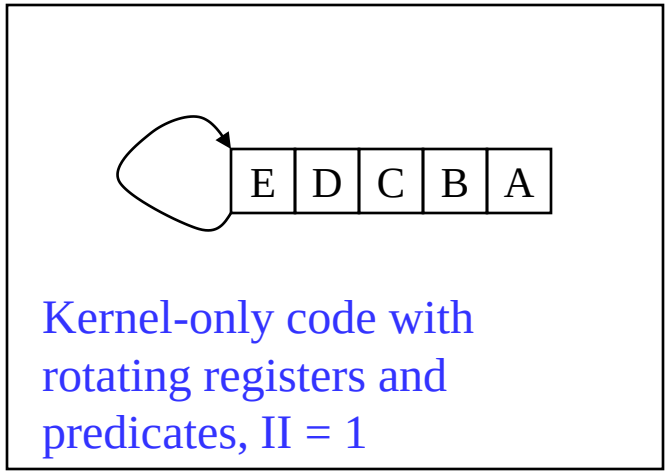
If the resources are non-pipelined, how many resources of each type are required to achieve II=1

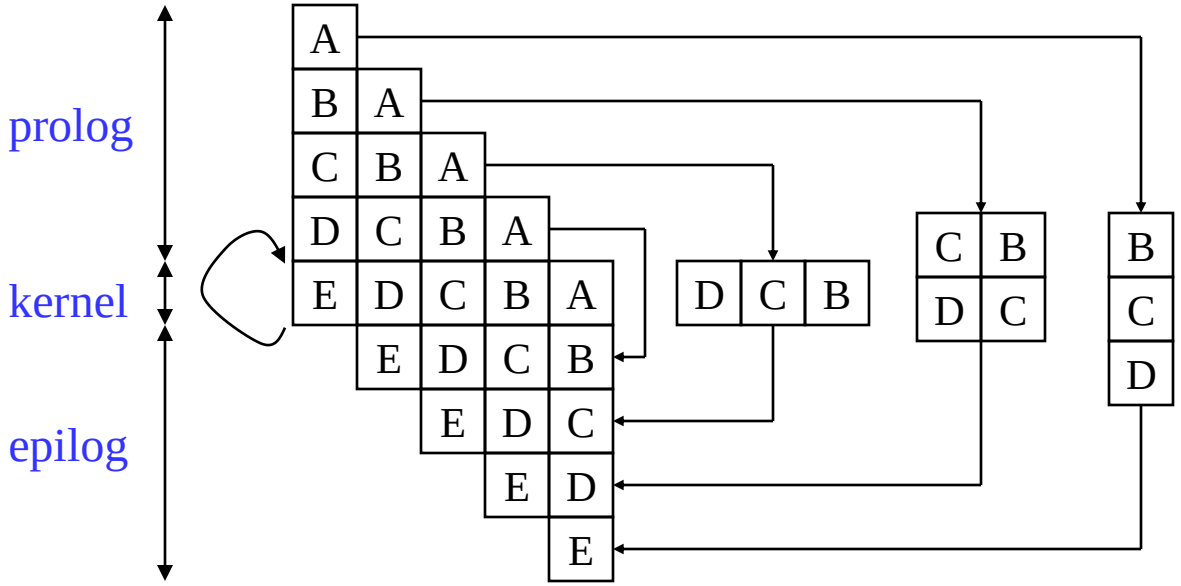
Assuming pipelined resources, generate the II=1 modulo schedule.

What if We Don't Have Hardware Support for Modulo Scheduling?

- No predicates
  - Predicates enable kernel-only code by selectively enabling/disabling operations to create prolog/epilog
  - Now must create explicit prolog/epilog code segments
- No rotating registers
  - Register names not automatically changed each iteration
  - Must unroll the body of the software pipeline, explicitly rename
    - Consider each register lifetime i in the loop
    - Kmin = min unroll factor = MAXi (ceiling((Endi – Starti) / II))
    - Create Kmin static names to handle maximum register lifetime
  - Apply modulo variable expansion

No Predicates





Without predicates, must create explicit prolog and epilogs, but no explicit renaming is needed as rotating registers take care of this

No Predicates and No Rotating Registers

Assume Kmin = 4 for this example