

ARSe Instruction Scheduler: Milestone 4

Dependency Checker Logic

A large part of our algorithm is the dependency checker, which compares two instructions and controls a dependency bit for the second instruction. It checks the register and memory usage between the two to determine if the second instruction can potentially be moved above the first. It handles special jump cases as well, making sure that jump destinations aren't relocatable, and that jump sources don't cross store boundaries.

We have created a table of the dependencies that extends the dependency checking portion of Milestone 3. The OpCodes for each instruction have been selected to exploit the similarity between dependencies. Each unique type of dependency has a letter associated with it, and you'll notice that several dependencies between unrelated instructions share the same letter, or type. We then grouped the dependencies by type and register usage to create the table on page 3. This allowed us to fairly easily create a set of equations describing the entire dependency checker. These equations can be plugged directly into a PLA generator to create the actual layout for our dependency checker subcell. The equations are formulated on page 3.

memop=01 1=store 0=load
unop=10
binop=11
jump=00 1=destination 0=source

	0	1	r1	m1	0	load sink, m1
a	0	1	r1	x	1	store sink , x
b	0	1	r1	x	0	load sink , x
c	0	1	x	m1	1	store, x, m1
d	1	0	r1	x	x	unop sink , x, x
e	1	0	x	r1	x	unop x, sink , x
f	1	1	r1	x	x	binop sink , x, x
g	1	1	x	r1	x	binop x, sink , x
h	1	1	x	x	r1	binop x, x, sink
	0	1	r1	m1	1	store source, m1
b	0	1	r1	x	0	load source , x
j	0	1	x	m1	0	load x, m1
c	0	1	x	m1	1	store x, m1
d	1	0	r1	x	x	unop source , x, x
f	1	1	r1	x	x	binop source , x, x
k	0	0	x	x	0	jump x (where current not equal latency instruction)
	1	0	r1	r2	x	unop sink, source, x
a	0	1	r1	x	1	store sink , x
e	1	0	x	r1	x	unop x, sink , x
l	1	0	r2	x	x	unop source , x, x
d	1	0	r1	x	x	unop sink , x, x
f	1	1	r1	x	x	binop sink , x, x
g	1	1	x	r1	x	binop x, sink , x
h	1	1	x	x	r1	binop x, x, sink
m	1	1	r2	x	x	binop source , x, x
n	0	1	r2	x	0	load source , x
b	0	1	r1	x	0	load sink , x
	1	1	r1	r2	r3	binop sink, source1, source2
a	0	1	r1	x	1	store sink , x
n	0	1	r2	x	0	load source1 , x
p	0	1	r3	x	0	load source2 , x
b	0	1	r1	x	0	load sink , x
d	1	0	r1	x	x	unop sink , x, x
f	1	1	r1	x	x	binop sink , x, x
g	1	1	x	r1	x	binop x, sink , x
h	1	1	x	x	r1	binop x, x, sink
m	1	1	r2	x	x	binop source1 , x, x
q	1	1	r3	x	x	binop source2 , x, x
e	1	0	x	r1	x	unop x, sink , x
l	1	0	r2	x	x	unop source1 , x, x
r	1	0	r3	x	x	unop source2 , x, x
	0	0	x		1	jd
s	x	x	x	x	x	any instruction
	x	x	x	x	x	any instruction
t	0	0	x		1	jd
	0	0	10000		0	j 10000
v	x	x	x	x	x	any instruction
w	0	1	x	x	1	store x, x
y	0	0	x		0	j x

	0	1	r1	m1	0
a	0	1	r1	x	1
b	0	1	r1	x	0
d	1	0	r1	x	x
f	1	1	r1	x	x
e	1	0	x	r1	x
g	1	1	x	r1	x
h	1	1	x	x	r1
c	0	1	x	m1	1
	0	1	r1	m1	1
b	0	1	r1	x	0
d	1	0	r1	x	x
f	1	1	r1	x	x
j	0	1	x	m1	0
c	0	1	x	m1	1
k	0	0	x	x	0
	1	0	r1	r2	x
a	0	1	r1	x	1
b	0	1	r1	x	0
d	1	0	r1	x	x
f	1	1	r1	x	x
l	1	0	r2	x	x
m	1	1	r2	x	x
n	0	1	r2	x	0
g	1	1	x	r1	x
e	1	0	x	r1	x
h	1	1	x	x	r1
	1	1	r1	r2	r3
a	0	1	r1	x	1
b	0	1	r1	x	0
d	1	0	r1	x	x
f	1	1	r1	x	x
n	0	1	r2	x	0
m	1	1	r2	x	x
l	1	0	r2	x	x
r	1	0	r3	x	x
q	1	1	r3	x	x
p	0	1	r3	x	0
g	1	1	x	r1	x
e	1	0	x	r1	x
h	1	1	x	x	r1
	0	0	x		0
s	x	x	x	x	x
	x	x	x	x	x
t	0	0	x		1
	0	0	10000		0
v	x	x	x	x	x
w	0	1	x	x	1
y	0	0	x		0

The equations are as follows:

- 1) a, b, d, f
 $a_0 * a_1 * a_7 * (b_0 + b_1) * (R_{1A} = R_{1B})$
- 2) b, d, f
 $a_0 * a_1 * a_7 * (b_0 * b_1 * b_7' + b_0) * (R_{1A} = R_{1B})$
- 3) a, b, d, f
 $a_0 * (b_0 * b_1 + b_0) * (R_{1A} = R_{1B})$
- 4) e, g, h
 $b_0 * (a_0 * a_1 * a_7' + a_0) * [(R_{1A} = R_{2B}) + (b_1 * (R_{1A} = R_{3B}))]$
- 5) j, c
 $a_0 * a_1 * b_0 * b_1 * (M_{1A} = M_{1B}) * [a_7 * + (a_7 * b_7)]$
- 6) k
 $b_0 * b_1 * a_0 * a_1 * a_7 * (CURR \neq LATENT)$
- 7) l, m, n
 $(b_0 + b_0 * b_1 * b_7') * a_0 * (R_{1B} = R_{2A})$
- 8) p, q, r
 $(b_0 + b_0 * b_1 * b_7') * a_0 * a_1 * (R_{1B} = R_{3A})$
- 9) s
 $a_0 * a_1 * a_7$
- 10) t
 $b_0 * b_1 * b_7$
- 11) v
 $a_0 * a_1 * a_7 * (MIN_DEST = 10000)$
- 12) w, y
 $a_0 * a_1 * a_7 * b_0 * ((b_1 * b_7) | (b_1 * b_7'))$

Note: CURR \neq LATENT, and (MIN_DEST=10000) are two bits that will be brought in externally by the algorithm. Likewise all of the register equality tests will be done external to the dependency checker.

If any one of these equations is true, then the dependency bit for the second instruction (whose bits are marked b_x) will be set. Thus the 9 equations above will fit in the AND stage of the PLA and then combined in the OR stage.