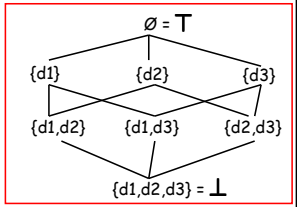
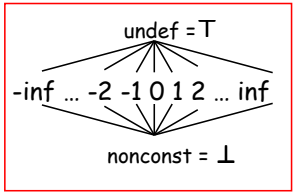
15-745 Lecture 4

Review
Big Picture
Pegasus Semantics
Pegasus Internals/Code
Assignment 1
C6X? Demo?

What the … is a Lattice?

- Represents values:
  for one item, or vector of all
  (often boolean to powerset)
- Has a defined top and bot
- According to ASU:
  - Top is least info:
    top ∧ X = X
  - Bot is end:
    bot ∧ X = bot
  - Init in[b] with top,
    out[b] with F_b(top).





Knowledge

``All I know is that I don't know …
all I know is that I don't know nothin"
- Operation Ivy
- as quoted by Green Day

That is, knowing that you know nothing is
more knowledge than not knowing
that you know nothing

Constructing Gen & Kill

Stmt s	gen[s]	kill[s]
t <- x op y	{x op y}-kill[s]	{exprs containing t}
t <- M[a]	{M[a]}-kill[s]	{exprs containing t}
M[a] <- b	{}	{for all x, M[x]}
f(a, …)	{}	{for all x, M[x]}
t <- f(a,…)	{}	{exprs containing t for all x, M[x]}

Constructing Gen & Kill

Stmt s	gen[s]	kill[s]
t <- x op y	{x op y}-kill[s]	{exprs containing t}
t <- M[a]	{M[a]}-kill[s]	{exprs containing t}
M[a] <- b	{M[a]}	{for all x, M[x]}
f(a, …)	{}	{for all x, M[x]}
t <- f(a,…)	{}	{exprs containing t for all x, M[x]}

Constructing Gen & Kill

Stmt s	gen[s]	kill[s]
t <- x op y	{x op y}-kill[s]	{exprs containing t}
t <- M[a]	{M[a]}-kill[s]	{exprs containing t}
M[a] <- b	{M[a]}	{for all x, M[x]}
f(a, …)	{}	{for all x, M[x]}
t <- f(a,…)	{}	{exprs containing t for all x, M[x]}

But…what if b gets overwritten?

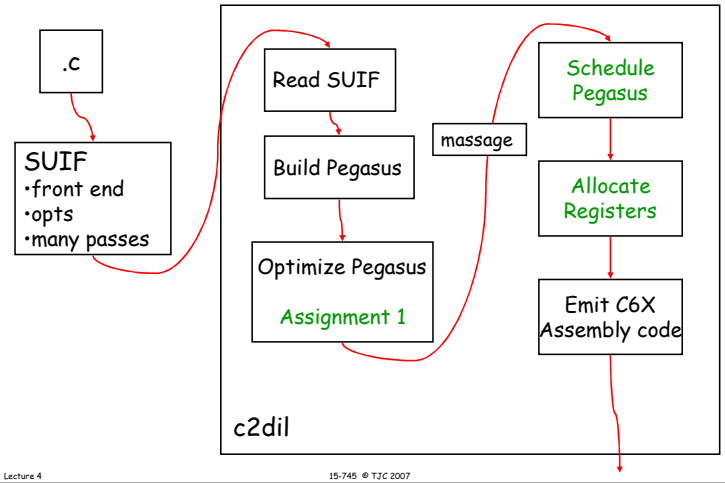
Constructing Gen & Kill

Stmt s	gen[s]	kill[s]
t <- x op y	{x op y}-kill[s]	{exprs containing t}
t <- M[a]	{M[a]}-kill[s]	{exprs containing t}
M[a] <- b	{M[a]}	{for all x, M[x]}
f(a, …)	{}	{for all x, M[x]}
t <- f(a,…)	{}	{exprs containing t for all x, M[x]}

But, how is a store an available expression?

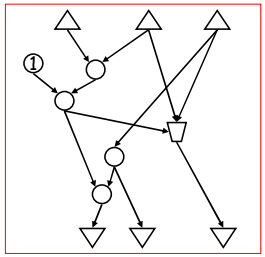
Big Picture

Big Picture



Big Picture

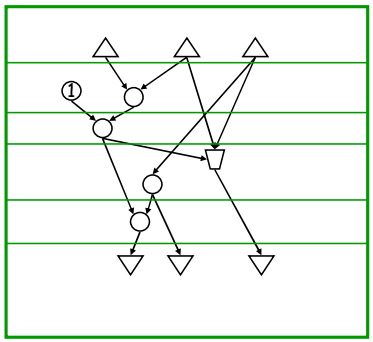
Schedule Pegasus



Big Picture

Schedule Pegasus

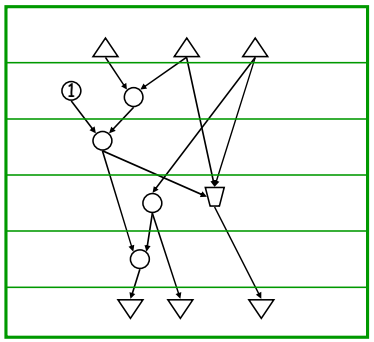
- users must come after producers
  - only independent ops can be scheduled in the same instruction



Big Picture

Schedule Pegasus

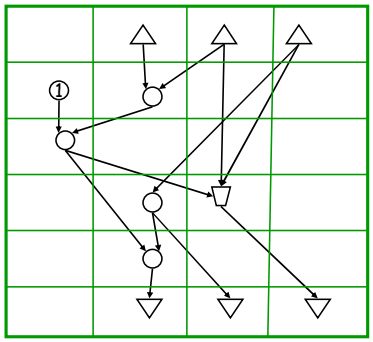
- users must come after producers
  - only independent ops can be scheduled in the same instruction



Big Picture

Schedule Pegasus

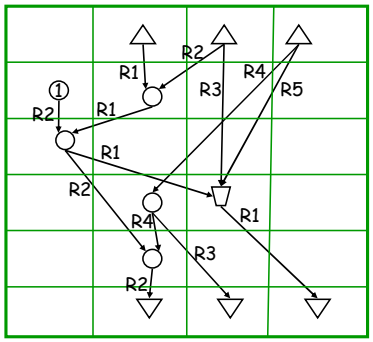
- ops must execute in a specific slot – on a specific function unit
  - resource conflicts might require ops to be delayed



Big Picture

Do register allocation

- If two spans overlap, they can't use the same register…
- End result: label each wire/edge with a register

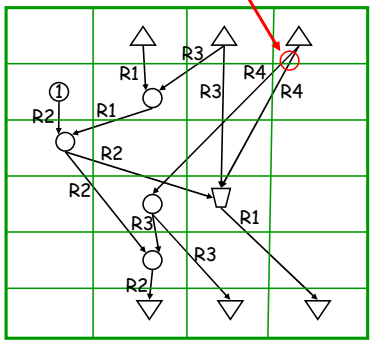


Big Picture

Do register allocation

- If two spans overlap, they can't use the same register…
- End result: label each wire/edge with a register

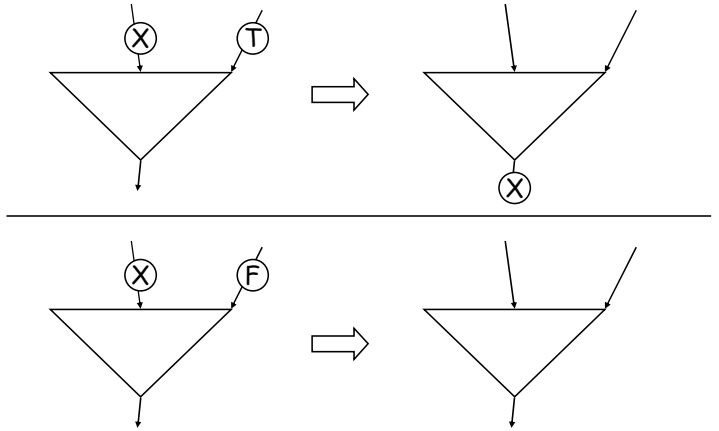
one wire w/ multiple dests



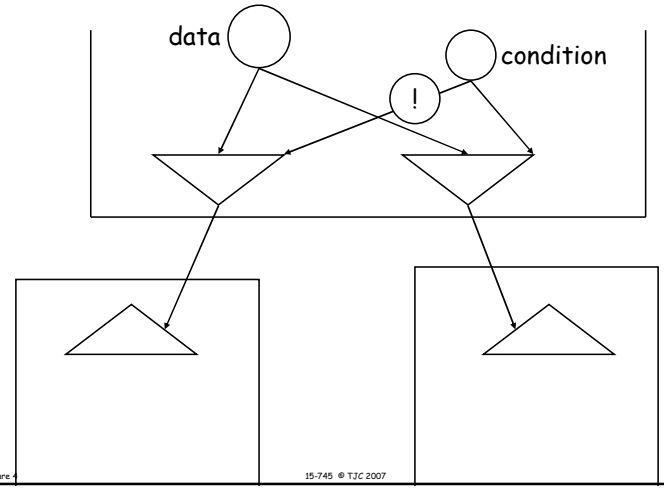
More Pegasus!

- etas/mus
- token edges
- multiplexors
- crt: current point
- other weird stuff

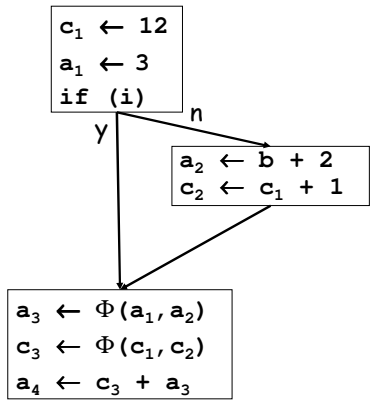
Pegasus: Etas (gateways)



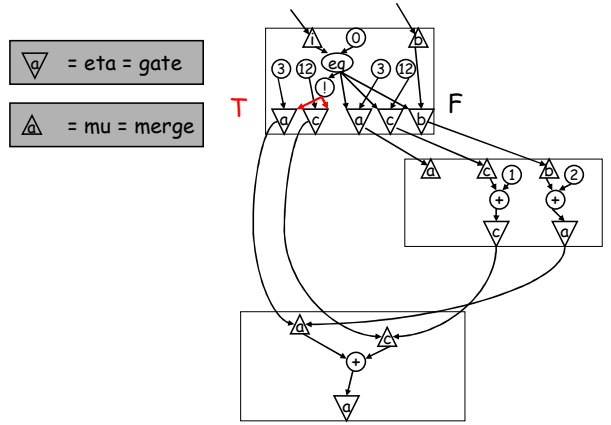
Pegasus: Etas (gateways)



Pegasus etas and mus



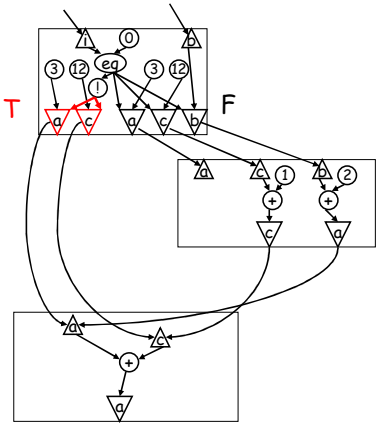
Pegasus etas and mus



Pegasus etas and mus

= eta = gate

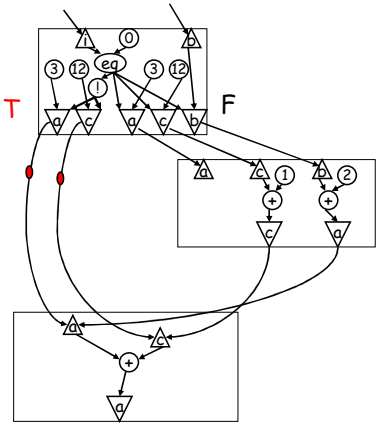
= mu = merge



Pegasus etas and mus

= eta = gate

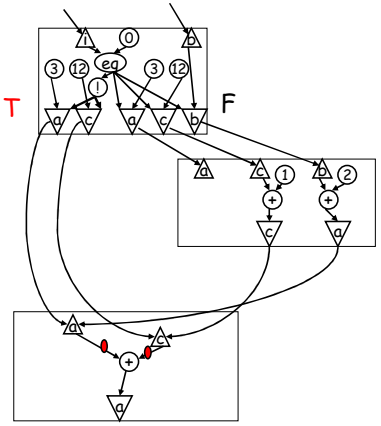
= mu = merge



Pegasus etas and mus

= eta = gate

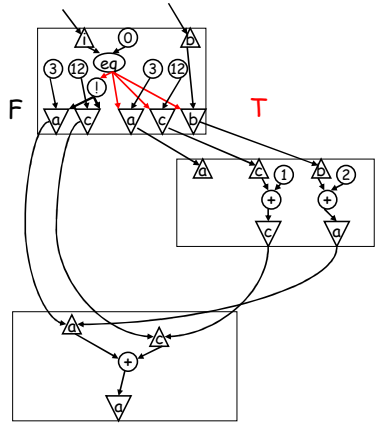
= mu = merge



Pegasus etas and mus

= eta = gate

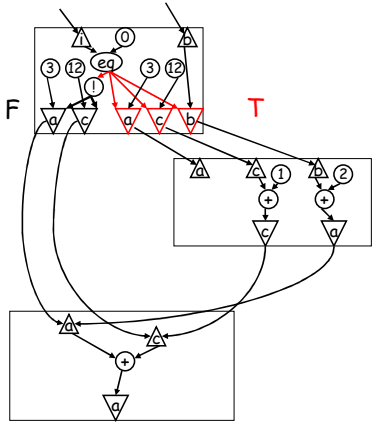
= mu = merge



Pegasus etas and mus

= eta = gate

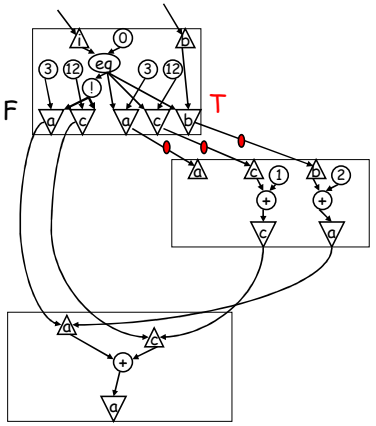
= mu = merge



Pegasus etas and mus

= eta = gate

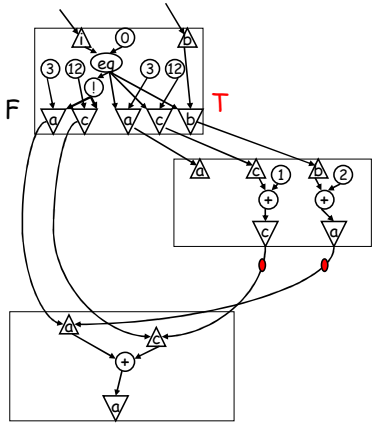
= mu = merge



Pegasus etas and mus

= eta = gate

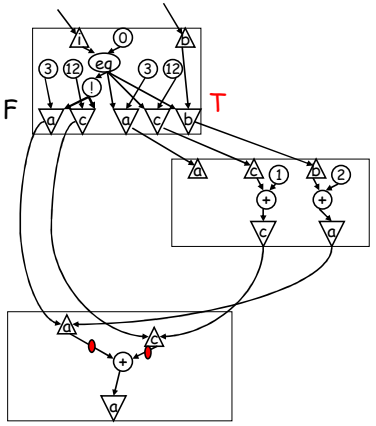
= mu = merge



Pegasus etas and mus

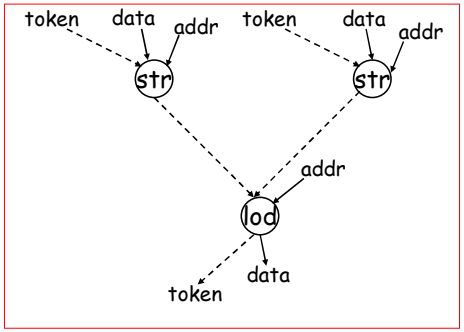
= eta = gate

= mu = merge



Pegasus token edges

- No data; they just enforce ordering between operations

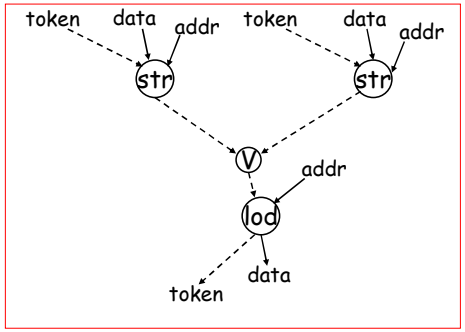


•they are also wires, and go through etas and mus…

•warning: inputs not shown in correct order

Pegasus token edges

- No data; they just enforce ordering between operations



•they are also wires, and go through etas and mus…

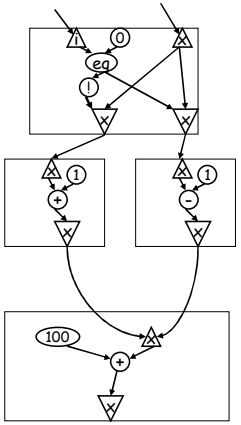
•warning: inputs not shown in correct order

•"token and" – a.k.a tkand

•and..predicates

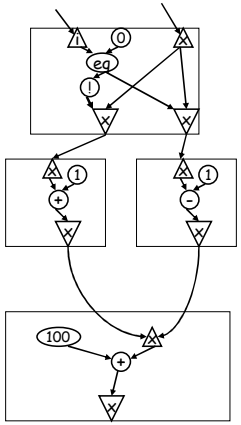
Pegasus multiplexors/predication

```
if (i)
   x++;
else
   x--;
x += 100;
```



Pegasus multiplexors/predication

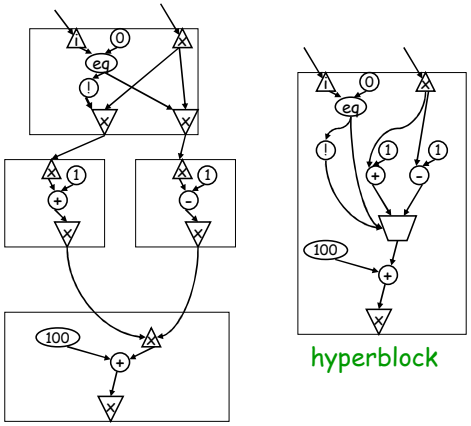
```
if (i)
   x++;
else
   x--;
x += 100;
```



If we go ahead and execute both paths, can we get rid of some overhead?

Pegasus multiplexors/predication

```
if (i)
   x++;
else
   x--;
x += 100;
```

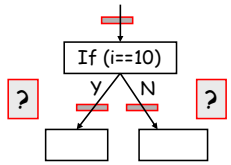


hyperblock

Assignment 1

Dataflow : Edges

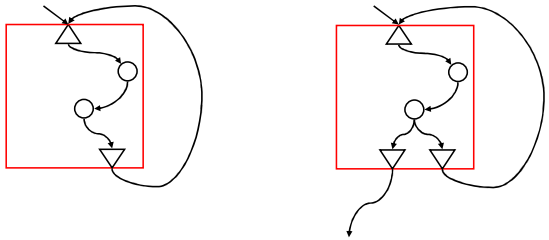
- Our framework supports differentiated information on the outgoing edges.



- Task 1A: add data information resulting from analysis of the conditionals to our existing conditional constant propagation pass

Aggressive DCE

- Task 1B: implement ADCE: assume everything dead unless proven live



Task 1B Alternatives

- Extend CCP+1A to range analysis:
  a value might not be constant, but is discovered to be bounded by a range
  - This is a somewhat open-ended problem, so talk to us before you go down this route
- ~~Perform induction variable based strength reduction in Pegasus~~

Admin

- TJC will have office hours tomorrow 10am-noon.
- Until I figure out 4-up landscape pdf handouts, you can look at the lectures from 15745-s05…