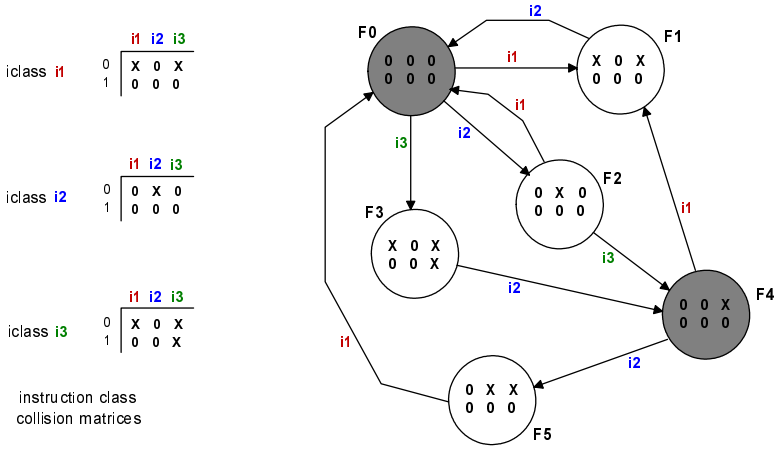
Forward Finite State Machine

```
ALGORITHM    Build_Forward_FSM

transition_table = state_hashtable = NULL;
new_states_list = F0;   /* initially contains the empty start state */
while (new_states_list is not empty) {
    Fa = remove first state from new_states_list;
    for (each instruction class i) {
        if (i is legal to issue from state Fa) {
            Fb = state created on an i transition from Fa;
            if (Fb does not exist in the state_hashtable) {
                add Fb to new_states_list;
                add Fb to state_hashtable;
            }
            add transition Fa --i--> Fb to the transition_table;   /* only legal transitions are recorded */
            issued_an_inst = TRUE;
        }
    }
    if (! issued_an_inst) {
        remove Fa from state_hashtable;
        shift its collision matrix to advance the pipeline cycle;
        mark Fa as cycle-advancing;
        if (the modified Fa does not exist in state_hashtable) {
            add modified Fa to new_states_list;
            add modified Fa to state_hashtable;
        }
    }
}
```


Forward Finite State Machine (example)



	F0	F4
F0	F0	F4
F4	F4	F4

Join Table

Factoring

- FSMs for complex processors can be large: e.g., 13,524 states for the Alpha 21064 !
- Solution: factor the FSM into multiple smaller FSMs.
- Factored FSM for Alpha has 237 + 232 states.

	0	1	2	3	4
int/nop0	ib0				
fp/nop1	ib1				
flt-st	ib0+l				
int-st	ib1+l				
loads	ibx+l				
imull	ib0+m	m^15	m+r	m^2	
imulq	ib0+m	m^17	m+r	m^2	
stx-c	ib1+l	1	1		
fdivs	ib1+f	f^24	f+x	f^4	
fdivq	ib1+f	f^52	f+x	f+x	f^4

{ib0,ib1,ibx,l,m,r}

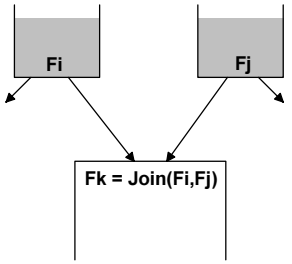
	0	1	2	3	4
int/nop0	ib0				
fp/nop1	ib1				
flt-st	ib0+l				
int-st	ib1+l				
loads	ibx+l				
imull	ib0+m	m^15	m+r	m^2	
imulq	ib0+m	m^17	m+r	m^2	
stx-c	ib1+l	1	1		
fdivs	ib1				
fdivq	ib1				

{ib0,ib1,ibx,x,f}

	0	1	2	3	4
int/nop0	ib0				
fp/nop1	ib1				
flt-st	ib0				
int-st	ib1				
loads	ibx				
imull	ib0				
imulq	ib0				
stx-c	ib1				
fdivs	ib1+f	f^24	f+x	f^4	
fdivq	ib1+f	f^52	f+x	f+x	f^4

The Join Table

- Resources can be in use past the end of a block.
- What state do we *begin* from at the top of a block?



- **Fi** and **Fj** must be cycle-advancing states.
- Join(**Fi**,**Fj**) is *at least* as restrictive as either **Fi** or **Fj**.
- Number of join states = $\frac{1}{2}$ (#cycle-advancing states)^2

- Modify Algorithm Build_Forward_FSM to construct the join states on the fly.
- Some join states may not have any inputs!
- Join states can be accessed using a *join table*.